(4)

VLSI Memo No. 88-465
August 1988

A MEMORY DESIGN FOR THE MESSAGE-DRIVEN PROCESSOR

Soha M. N. Hassoun

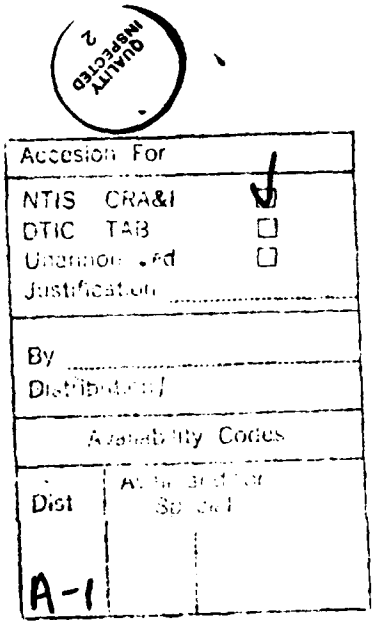
Abstract

The Message-Driven Processor (MDP) is a low-latency processing node for a scalable fine-grain MIMD concurrent computer, the Jellybean Machine. Programs are executed by passing messages through a low-latency network. Each MDP integrates a processor, a memory, and a communication network. On top of this message-passing model, the MDP supports a global virtual address space.

This thesis involves the design and implementation of a memory for the Message-Driven Processor. The memory array can be accessed by index, by row, or as a set-associative cache. Index operations are used to read and write memory. Row operations reduce the latency in message-handling by providing special purpose buffers, Row Buffers that access four words (a row) of memory simultaneously. Two Queue Row Buffers enable buffering messages at two different priority levels as soon as they arrive from the network. An Instruction Row Buffer acts as a small instruction cache. Set-associative operations provide a translation mechanism to enable translating any object to its associated item. MDP operating system routines use this cache to translate virtual identifiers into global addresses.

The microarchitecture and the circuit design of the memory is developed. A test chip is fabricated to verify the design. Evaluation of the row operations is presented.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138



Acknowledgements

Submitted to the Department of Electrical Engineering and Computer Science on May 13, 1988 in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering and Computer Science. This work was supported in part by the Defense Advanced Research Projects Agency under contract nos. N00014-80-C-0622 and N00014-87-K-0825.

Author Information

Hassoun: Digital Equipment Corporation, Mail Stop HLO2-3/c12, 77 Reed Road, Hudson, NY 01749-2809.

A MEMORY DESIGN FOR THE MESSAGE-DRIVEN PROCESSOR

by

Soha M.N. Hassoun

B.S.E.E., South Dakota State University
(1986)

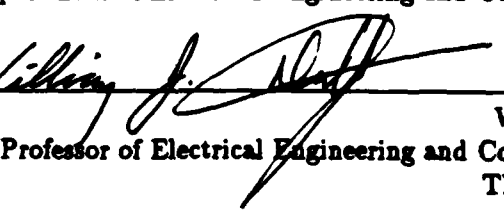
SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS OF THE
DEGREE OF

MASTER OF SCIENCE
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 1988

Signature of Author 
Department of Electrical Engineering and Computer Science
May 13, 1988

Certified by 
William J. Dally
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by _____
Dr. Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

A MEMORY DESIGN FOR THE MESSAGE-DRIVEN PROCESSOR

by

Soha M.N. Hassoun

Submitted to the
Department of Electrical Engineering and Computer Science
on May 13, 1988 in partial fulfillment of
the requirements for the Degree of Bachelor of Science in
Electrical Engineering and Computer Science

Abstract

The Message-Driven Processor (MDP) is a low-latency processing node for a scalable fine-grain MIMD concurrent computer, the Jellybean Machine. Programs are executed by passing messages through a low-latency network. Each MDP integrates a processor, a memory, and a communication network. On top of this message-passing model, the MDP supports a global virtual address space.

This thesis involves the design and implementation of a memory for the Message-Driven Processor. The memory array can be accessed by index, by row, or as a set-associative cache. Index operations are used to read and write memory. Row operations reduce the latency in message-handling by providing special purpose buffers, Row Buffers that access four words (a row) of memory simultaneously. Two Queue Row Buffers enable buffering messages at two different priority levels as soon as they arrive from the network. An Instruction Row Buffer acts as a small instruction cache. Set-associative operations provide a translation mechanism to enable translating any object to its associated item. MDP operating system routines use this cache to translate virtual identifiers into global addresses.

The microarchitecture and the circuit design of the memory is developed. A test chip is fabricated to verify the design. Evaluation of the row operations is presented.

Thesis Supervisor: William J. Dally
Title: Assistant Professor of Electrical Engineering and Computer Science

Keywords: DRAM, Cache, Row Buffers, Jellybean Machine.

Acknowledgements

I would like to thank my parents for their *infinite* supply of love, support and understanding. *Thank you* for standing by me during the hard times. Words fall short describing my love, respect, and gratitude.

Special thanks to my brother, Marwan, for his technical and nontechnical advice. *Thank you* for making my world a happier place.

Thanks to every member of the teaching staff at South Dakota State University, specially Dean Earnest Buckley, for their continuous support and encouragement to attend graduate school. *Thank you.*

Mega-thanks to my thesis advisor, Bill Dally, for giving me a chance to work on his Jellybean machine. His knowledge and ideas made this thesis possible. *Thank you*, Bill.

I would like to thank all the members of the CVA group for their support. *Thank you* Stuart Fiske for helping me think out a lot of circuit details, your patience, and for being a great office-mate. *Thank you* Andrew Chien for reading parts of my thesis and for the helpful comments during the last year. *Thank you* Jerry Larivee for your help around the lab. Special thanks to you, Brian Totty, for the time that you spent trying to make me understand "MDP stuff" and the constant encouragement to finish my chip. Also thank you for you photographic expertise in taking the pictures in Chapter 5 and the formatting of the quotes and the equations all over.

I would also like to thank Prof. John Wyatt, John Wade, Silvano Brewster for interesting informative discussions and for answering many of my questions.

To my parents, Marwan, and Marwa.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *I would have you imagine, then, that there exists in the mind of man a block of wax, which is of different sizes in different men; harder, moister, and having more or less of purity in one than another, and in some an intermediate quality. . . . Let us say that this tablet is the gift of Memory.*

— PLATO, in *Dialogues, Parmednides, p. 191*

The Jellybean Machine is a fine-grain concurrent machine that supports an object-oriented programming model. Programs compute in a message-passing style. Computing nodes are configured in a 2-dimensional grid. Each single-chip node integrates a communication network, a processor, and a memory. A processor is either a symbolic processor or an object expert that performs operations on certain types of objects. The Jellybean Machine is currently being developed by the Concurrent VLSI Architecture (CVA) group at MIT under the supervision of Professor William Dally.

The Message-Driven Processor (MDP) [D*87] is the symbolic processing node for the Jellybean machine. The message-handling overhead on a node is reduced by providing hardware support to buffer and execute messages and to switch context rapidly. Messages are buffered in the on-chip memory as soon as they arrive from the network. They are executed through direct interpretation instead of the fetch-decode-execute loop of conventional

processors. Fast context switching is supported by providing two sets of processor registers and two message queues.

On top of this message-passing model, the MDP supports a global virtual address space. The on-chip memory can be accessed as a set-associative cache to translate a virtual address into its physical address.

1.1 Focus

This thesis focuses on the design of the memory for a prototype Message-Driven Processor. Unlike conventional memory organizations that access separate Random Access Memories (RAMs) and caches, the MDP uses one physical memory structure that can be accessed by an index to read/write a single word, or as a set-associative cache to translate a key into its associated value.

In addition, the MDP memory implements *row operations* to accomplish:

1. Buffering incoming messages from the network at two different priority levels to reduce the total memory cycles needed to store messages in memory.
2. Providing fast access to the instruction stream by fetching 8 instructions from memory at a time.

A test chip was fabricated to evaluate this design and the *row operations* preformance was analyzed.

1.2 Background: The MDP and the Jellybean Machine

1.2.1 Execution Model

The Message-Driven Processor transmits and executes messages at two priority levels. The message header is the x and y-coordinates of the message's destination node. The on-chip communication networks routes a message to it's final destination without disrupting the processors. At it's final destination, the message header is stripped off and the message is buffered in one of the queues in memory according to its priority level. The queues are circular FIFO buffers that hold the messages to be executed. The processor executes the message at the head of the higher priority non-empty queue. If both queues are empty the processor is in an idle state.

Messages consist of the message opcode followed by the message's arguments. Message opcodes are physical addresses of routines that support the object-oriented programming model, code execution, storage allocation, and various other utilities. Frequently used routines reside in the on-chip ROM. Once the appropriate routine is executed, a message is dequeued from memory and the processor executes another message.

A translation look-aside buffer (TLB) is used to lookup any type of data associated with a certain key. The message routines use this TLB to translate an object's global identifier (ID) into its physical address and to lookup the method that is associated with a class/selector pair.

1.2.2 Architecture

The MDP consists of the Address Arithmetic Unit (AAU), the Register Arithmetic and Logical Unit (RALU), the Control Unit (CU), and the Memory Unit (MU). These units are connected through buses and some global signals. Each processor is connected to other processors through a low-latency network [DS87].

The AAU calculates an address to access the memory. It performs several functions to support enqueueing, dequeueing and dispatching messages that arrive from the network. It also supervises the instruction pointer, the stack pointer, and some status bits. The RALU contains the register file and the hardware to perform logical and arithmetic operations on data stored in the registers. It checks the type and range of arithmetic operations. The CU fetches instructions from the instruction cache. It decodes and pipelines the instruction stream into several commands and broadcasts them to the appropriate units. In addition, it monitors and handles all the faults and traps generated by the other units.

The Memory Unit (MU), which is the focus of this thesis, provides storage for the data and messages arriving from other processors. The unique organization of the memory allows it to be accessed by index or as a set-associative cache. The Memory Unit supports instruction fetching and message enqueueing by providing row operations that access four words (a row in the memory array) simultaneously.

1.2.3 Performance

The MDP handles a message dispatch and switches context within 5 μs. This low latency in message handling allows concurrent algorithms to be supported at their natural grain size of about 20 instructions.[Dal]

The prototype MDP will perform at 4 MIPS with a 36K-bit memory. The MDP will be fabricated using a 2 μm standard MOSIS process. The prototype Jellybean Machine will consist of 4K nodes, with 2K Message-Driven Processors and 2K numerical object experts (Reconfigurable Arithmetic Processor) [FD]. This machine will achieve 2G PTPS (pointer traversals per second) and 2G FLOPS (floating point operations per second)[DL87]

An industrial version of the MDP will have a memory capacity of 4K words. A full scale Jellybean machine will consist of 64K nodes, and its performance will scale accordingly.

1.3 Literature Survey

1.3.1 The Evolution of Memories

In the past 40 years, computers have used a variety of memories such as delay lines, magnetic drums, cathode-ray-tube storage, magnetic cores, magnetic film memories, semiconductor memories, charge-coupled devices and magnetic bubbles. The driving force behind this development is the need for increased density and speed and minimum power consumption. For example, in the last 25 years at IBM, memory density has increased 280,000 times, speed has improved 10-100 times, and power consumption per bit has decreased 20,000 times [P*81].

Semiconductor memories include Random Access memories (RAM) and read only memory (ROM). Random Access Memory implies that each location can be read or written with equal access time. In ROMs, binary information is easily read out but is written either permanently at fabrication time or electrically by the user.

The first commercial semiconductor memory was used in the IBM System/360 Model 85. In MOSFET memories, each bit is stored on a capacitance. Early memory cells were static made of a set of cross-coupled inverters to store the information and pass gates to access it. To reduce the CMOS cell size and dynamic power dissipation, the p-channel transistors were eliminated producing the four-transistor cell. The three-transistor cell eliminated the feedback loop within the cell and stored charge on a capacitance. Single transistor DRAM cells employ one transistor to access the storage capacitance. Figure 1.1 illustrates the evolution of memory cells.

The first commercial MOS DRAM was Intel's 1103. It was 1K words by 1 bit array. Processing technologies has allowed memory densities of 1M bits and experimental 4M and 16M bits per chip. Scaling feature sizes to less than 1 μm and using additional layers of polysilicon have reduced the size of the 1-transistor DRAM cell. Minimum DRAM cell sizes are achieved through Surrounding Hi-Capacitance cell structures, in which the side-walls

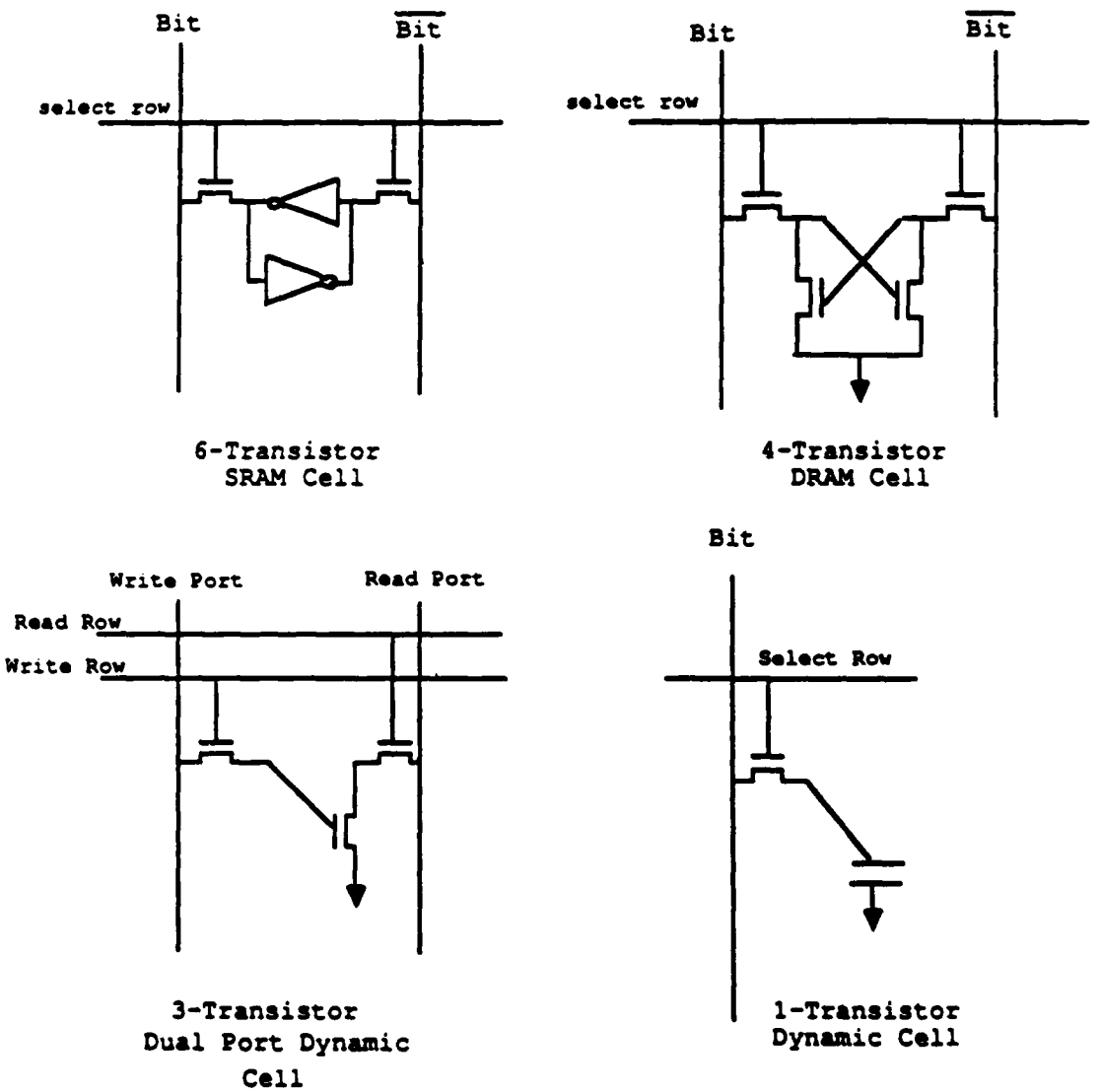


Figure 1.1: Memory Cell Evolution

capacitance of the trench form the storage capacitor. Matsushita Semiconductor Research Center, Osaka, Japan, reported a cell size of 1.5μm by 2.2μm, and a trench depth of 2.5μm using a 0.5μm N-well CMOS process [I*88]. The decrease in cell size has caused an increase in the soft error rate, the inter-bit line and bit line/word line coupling noise. Layout was strained by cell/sense amplifier pitch matching. Several memory array organizations such as folding and twisting the bit lines and dividing the array into several independent blocks have reduced these problems.

1.3.2 Advances in Cache Organizations

Besides technology advances, new memory organizations made memory access faster. "Look-aside buffers", fast registers that stored recently accessed data, were first were first introduced by Leon Bloom in 1962 [BCP62]. These fast local memories, or caches, were first commercially introduced by IBM in their System/360 model 85. The cache size ranged form 16 to 32 Kbytes. Also, some computer organizations provide two different caches, an instruction cache and a data cache.

1.4 Design Constraints

The prototype MDP will be fabricated using a standard 2μm double-metal CMOS MOSIS (MOS Implementation Service) process. The available technology and the size of available chips (7900μm x 9200μm) constrained the memory's basic cell design and the size of the memory array.

Our RAM design uses the 3-transistor memory cell. This cell occupies more area than a 1 transistor DRAM cell with the same storage capacitance using the same process. However, it is a more conservative design considering the variations within MOSIS processes. The size of the memory array was constrained to 1K words (36 bits/word) to fit on chip along with the rest of the MDP.

1.5 Summary

This thesis reports the design of the Memory Unit for the Message-Driven Processor. It includes the design of the memory and the fabrication of a test chip. It evaluates the implementation of the queue row buffers, the instruction row buffers, and the hardware support of the address translation mechanism.

Chapter 2 of this thesis is a description of the MDP's memory microarchitecture. In chapter 3, I describe the hardware design of the memory system, and in chapter 4, I describe the memory test chip. Chapter 5 is an evaluation of two architectural features of the MDP memory: the queue buffers and the instruction buffer. Chapter 6 is a summary and some suggestions to improve this memory design.

Chapter 2

Memory System Microarchitecture

'Tis in my memory locked,
And you yourself shall keep the key of it.

— SHAKESPEARE, in *Hamlet, I, iii, 75*

The MDP's Memory Unit (MU) provides storage for objects and messages. The memory array can be accessed by index or as a set-associative cache. The Memory Unit's microarchitecture optimizes writing new messages into memory by using two row buffers, the Queue Row Buffers (QRBs), to transfer a row (4 words) into memory simultaneously. Fetching instructions is optimized by fetching a row (8 instructions) from memory at once and storing it in an Instruction Row Buffer (IRB).

This chapter describes the Memory Unit's functionality, its interface with the other MDP units, and its internal elements.

2.1 Functionality

The Memory Unit executes the operation specified by the MDP's Control Unit. The *read* and *write* decoded instructions load and store a data word from/to memory respectively. Four other decoded instructions access the memory as a set associative cache. The *xlate* instruction translates a key into its associated entry. If a cache miss occurs, a fault handler is invoked. The *probe* instruction checks if a certain key is present in the cache and returns a boolean value indicating if the key was found. The *enter* instruction writes a key and an associated item into the cache and the *purge* instruction deletes them.

The Memory Unit performs row operations to increase the memory's bandwidth. The Memory Unit fetches a row of the memory array and writes it into a special buffer, the Instruction Row Buffer (IRB). This buffer acts as an instruction cache that holds the next instructions to be executed. The MDP Control Unit initiates the fetching operation as necessary. The Memory Unit enqueues messages arriving at the Network Unit by first buffering them in one of the Queue Row Buffers (QRBs), and then writing that buffer into memory. The QRBs are loaded into memory when they are full or if the last word of the message has arrived from the Network Unit.

Frequent refreshing of the memory array is necessary to restore the charge in the memory cells. The refresh operation has the highest priority, followed by writing the queues and finally, the execution of one of the decoded memory instructions and the loading of the IRBs.

2.2 Interface

The interface between the MU and other MDP units is necessary to specify the operation to be performed, the information (data, instructions, or messages) to be accessed , and the location in the memory array where information will be accessed. Global clocks and signals are used to synchronize this interface. (i.e. how, what, where, and when to access the

memory array.) The Memory Unit interacts with other MDP units as shown in Figure 2.1

The MDP Control Unit specifies a memory operation while decoding the instruction stream. Operations are either word operations, associative operations or row operations.

Information used in memory operations is either data, instructions or messages. The RALU transfers data between memory and one of its register through the data bus, the C-bus, prior to or after performing an arithmetic operation on it. The CU executes one of eight instruction in the Instruction Row Buffer at a time. The NU writes a word of a message into one of a slot in the QRBs.

The AAU generates a word address when storing or loading a word, and a row address when performing an associative operation or a row operation. Word addresses are generated by adding an offset to a segment base address stored in an AAU register. The AAU uses a Translation Base/Mask register (TBM) to hash a key into a row address where the key's associated items reside. It uses a Queue Head and Length Register (QHL) to generate a row address where the queue buffers are stored. A refresh counter in the AAU points to the address of the next row to be refreshed.

The MDP uses a two phase nonoverlapping clock as shown in in the top of Figure 2.2. Memory reads and writes are executed in one clock cycle. Since the AAU decodes and drives the address to the Memory Unit, and since information is transferred between MDP units in synchronization with the MDP's pipelined Control Unit, the execution of some decoded commands take several cycles. The memory executes the *write, enter, purge* decoded commands and row operations in one cycle, the *read* decoded command in two cycles, and the *xlate* and *probe* commands in 3 cycles. Figure 2.2 is a summary of the timing for each operation providing no interrupts are generated during execution. In the figure, *WData* refers to data to be written into the memory, *RData* refers to data to be read out of the memory, and *BData* is a boolean value.

To insure nonconflicting operations and to regulate the interface of the memory with all other MDP units, the MU asserts the following signals as necessary:

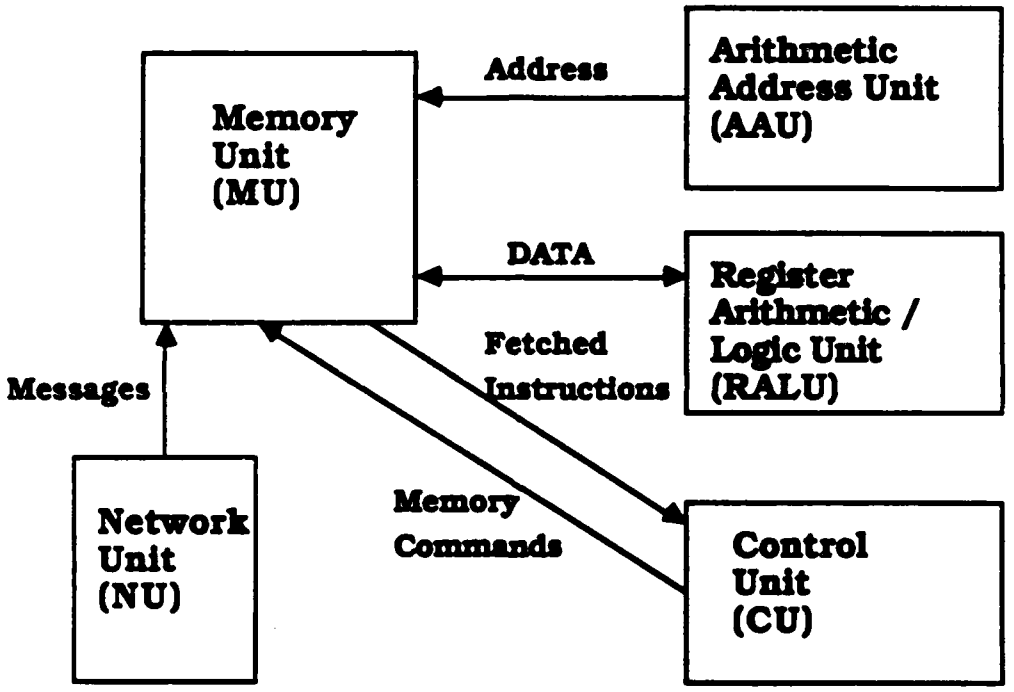


Figure 2.1: Interface between Memory Unit and other MDP Units

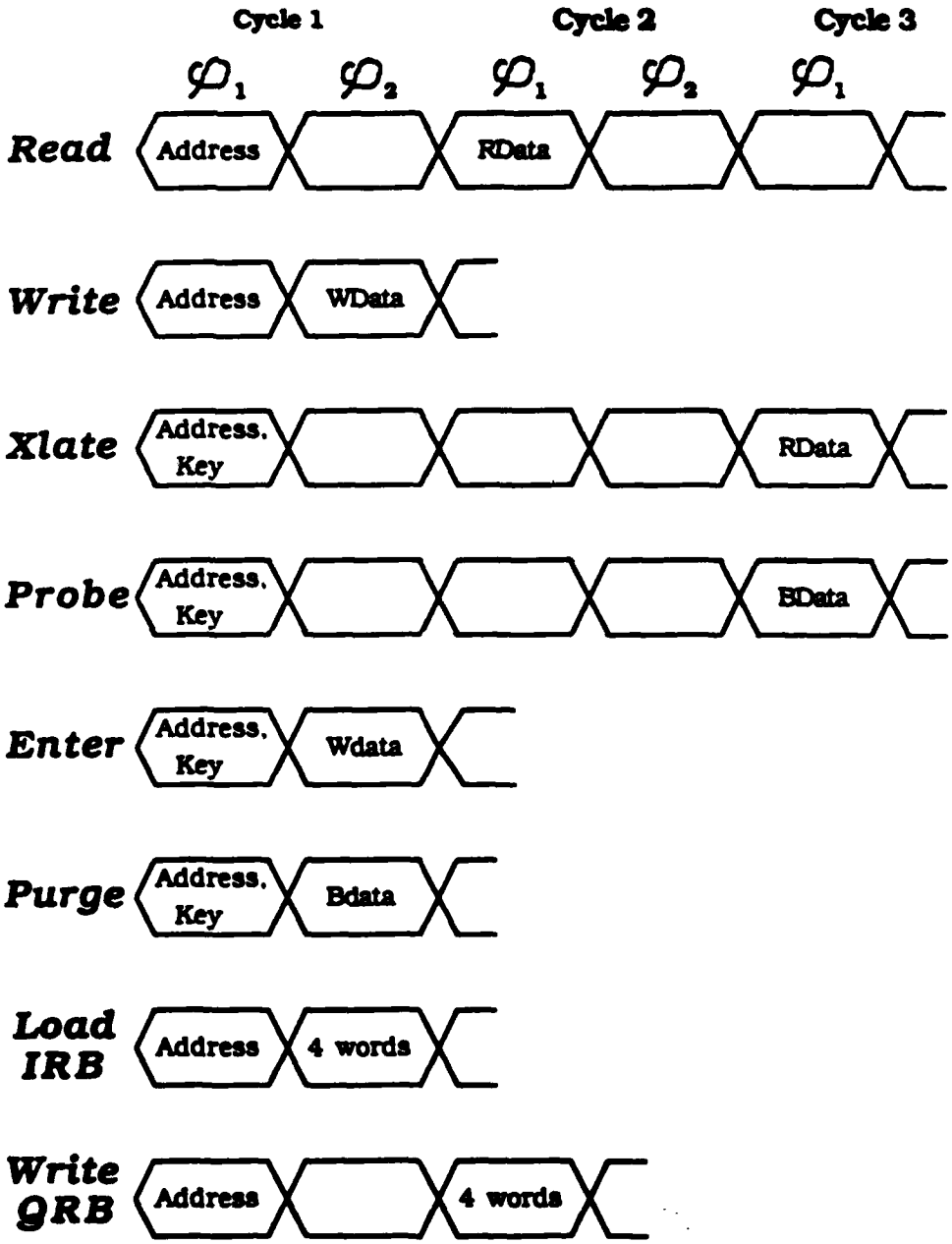


Figure 2.2: Timing of Memory Operations

1. The Ready Signal : A global wait signal that stalls the execution of the pipelined instruction stream. It is asserted when performing operations that require more than one cycle or when a refresh request *or*/and a queue buffer write request takes priority over the execution of the current memory command.
2. The MDR Valid Signal: A signal that indicates that data stored in the Memory Data Register is valid and is transferred on the C-bus.
3. The Squish Signal: A global control signal that halts any action that might change the state of the memory or processor.
4. The Memory Trap signal: This is generated in case of an unsuccessful *xlate* operation.
5. The Parity Trap signal: This signal is asserted in case of an uncorrectable parity error in reading the memory.

2.3 Memory Unit Elements

The functional block diagram of the Memory Unit is shown in Figure 2.3 A register-transfer level simulation of the Memory Unit was developed as part of the MDP. The following is a brief description of all elements of the Memory Unit. Appendix A describes this simulation in more detail.

2.3.1 Memory Controller

The local memory controller supervises all activities with other MDP units and within the Memory Unit.

The Memory Controller receives commands from the MDP Control Unit. It arbitrates between the decoded memory commands, the queue write requests, and the refresh requests and sends the result of the arbitration to the AAU to generate the appropriate address. It generates local memory commands to organize the data traffic between the C-bus, the

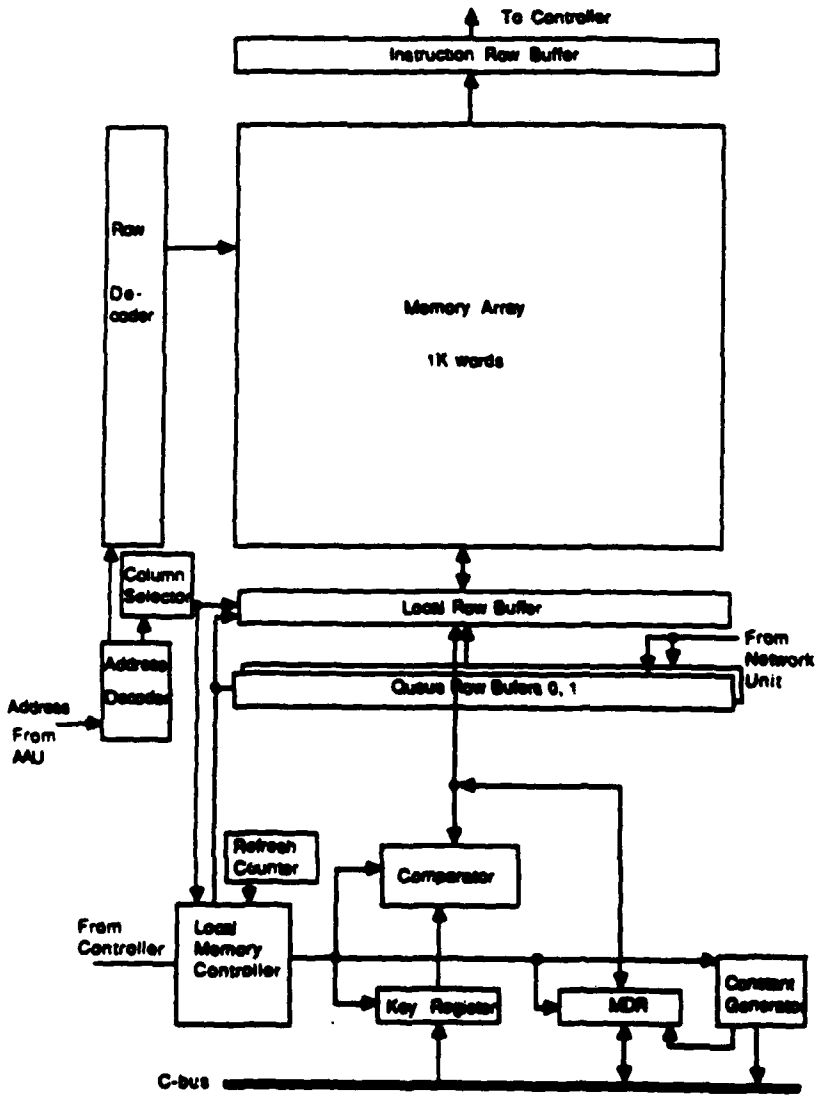


Figure 2.3: Memory Unit Functional Block Diagram

registers, and the local buffers. It also asserts the global signals when appropriate and organizes the enqueueing operation with the NU.

2.3.2 The Local Row Buffer

On one phase of the clock, a row of memory is loaded into the local row buffer. On the other phase, the local row buffer is written into the same memory row. Data in the local row buffer is written to the Memory Data Register and is used to perform the associative lookup. To enter data into memory, the contents of the local row buffer are modified before writing the local row buffer back into memory. Writing back the original contents of the row constitutes a refresh operation.

2.3.3 Memory Unit Registers

The memory has two different registers that are used to load/store data from/to memory. The Key Register holds the key to be translated The Memory Data Register holds the word to be written into the memory array when accessing the memory by index and the associated item when using the memory as a cache. It obtains data from the local row buffer and enables it on the C-bus when doing a read or a successful xlate.

2.3.4 Comparator

The comparator compares the data in the Key Register with the even words in the local row buffer when executing an associative command. Two HIT lines, which are active high, reflect the result of this comparison. If both HIT lines are discharged while executing an associative operation, the MU asserts the Memory Trap signal. Figure 2.4 illustrates this operation.

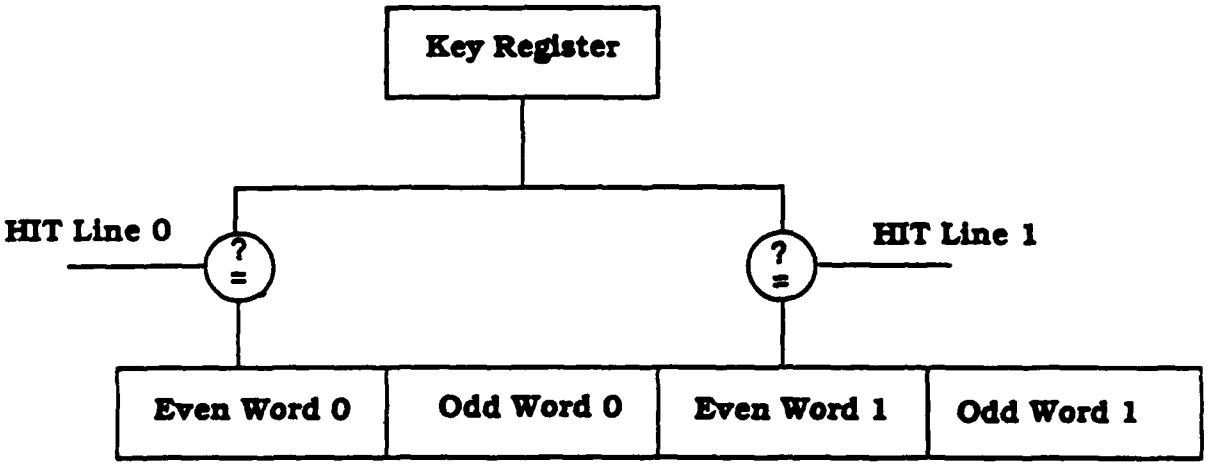


Figure 2.4: The Compare Operation

2.3.5 The Address Decoders and the Column Selector

The address decoders and the column selector decode the 10-bit address from the AAU into a memory location. The 8 higher bits of the address access one of the memory's 256 rows, while the lower 2 bits select one word out of four in that row when doing a word operation.

2.3.6 The Row Buffers

Row buffers speed up the execution of the instruction stream by reducing the memory cycles required to write the queues or to fetch the next instruction from memory. The two queue buffers hold words of a message before writing them into memory. The instruction row buffer fetches a row of the instructions in memory.

Analysis of the row buffers' performance is covered in Chapter 5.

2.4 Summary

At any point in time, the Memory Unit performs a word operation such as storing/loading a word of data in memory, or a row operation such as loading instructions from memory into the Instruction Row Buffer, writing the Queue Row Buffers, or refreshing a row in memory. The MDP's Control Unit specifies the operation to be performed, and the AAU generates the appropriate address. The Network Unit writes messages to the queue buffers and the Control Unit executes the fetched instructions in the IRB. All the interface is synchronized using global clocks and signals.

A register-transfer level simulation of the MDP has been developed by the CVA group at MIT to verify the microarchitecture. Appendix A describes the logic equations that describes a register transfer-level simulation of the MU.

Chapter 3

Memory Design

. . . . we hold the wax to the perceptions and thoughts, and in that material receive the impression of them as from the seal of a ring; and that we remember and know what is imprinted as long as the image lasts . . .

— PLATO, in *Dialogues, Parmednides, p. 191*

The MDP memory is a 36K bit (36 bits/word) array. It is arranged in 256 rows by 144 columns. Each memory cell is a 3-transistor DRAM cell. Bit lines are precharged high before reading the memory cells. The higher order bits of the address select a row in the memory array. The lower order bits of the address select a word in memory when reading or writing a word in memory. Comparators in the peripheral circuitry compare the two even words in a selected row against a key when performing a set-associative operation. The result of this comparison specifies the word(s) in the selected row to be accessed. Three row buffers, an Instruction Row Buffer and two Queue Row Buffers, enable simultaneous access to four row-aligned words of memory. A sense amplifier speeds up sensing the discharging of the bit lines when reading the memory. The memory was fabricated using a 2μm CMOS double-metal layer process. It is designed to run at 15.5MHz.

This chapter describes the timing and the circuit design of the MDP memory. More

details are provided in Appendices B and C.

3.1 Timing

The timing diagram of the memory is shown in Figure 3.1. The memory executes a write-precharge-read operation every clock cycle. The duration of φ_1 is limited by the time to perform a compare followed by a write operation. φ_2 is limited by the time to perform the precharge and the read operations.

The address is decoded by the falling edge of φ_1. Precharging the bit lines and selecting a row in the memory array occur simultaneously during φ_2. The precharge clock is low for 6.4 ns. The *row read signal* has a rise time of 5 ns. The selected row is read from the memory array by the falling edge of φ_2.

The *row write signal* is set high during the first 5 ns of the following φ_1. The result of the compare operation is valid 12.5ns after the rising edge of φ_1. Column drivers write data back into the selected row after the necessary modification during the remainder of φ_1. A new address and operation are decoded while completing the write cycle of the previous operation.

3.2 Circuits

Figure 3.2 is a functional block diagram of the memory's circuits. Appendix B contains schematics illustrating a vertical slice through Figure 3.2 and the details of the control circuitry. This section describes the circuit elements that determine the memory's performance.

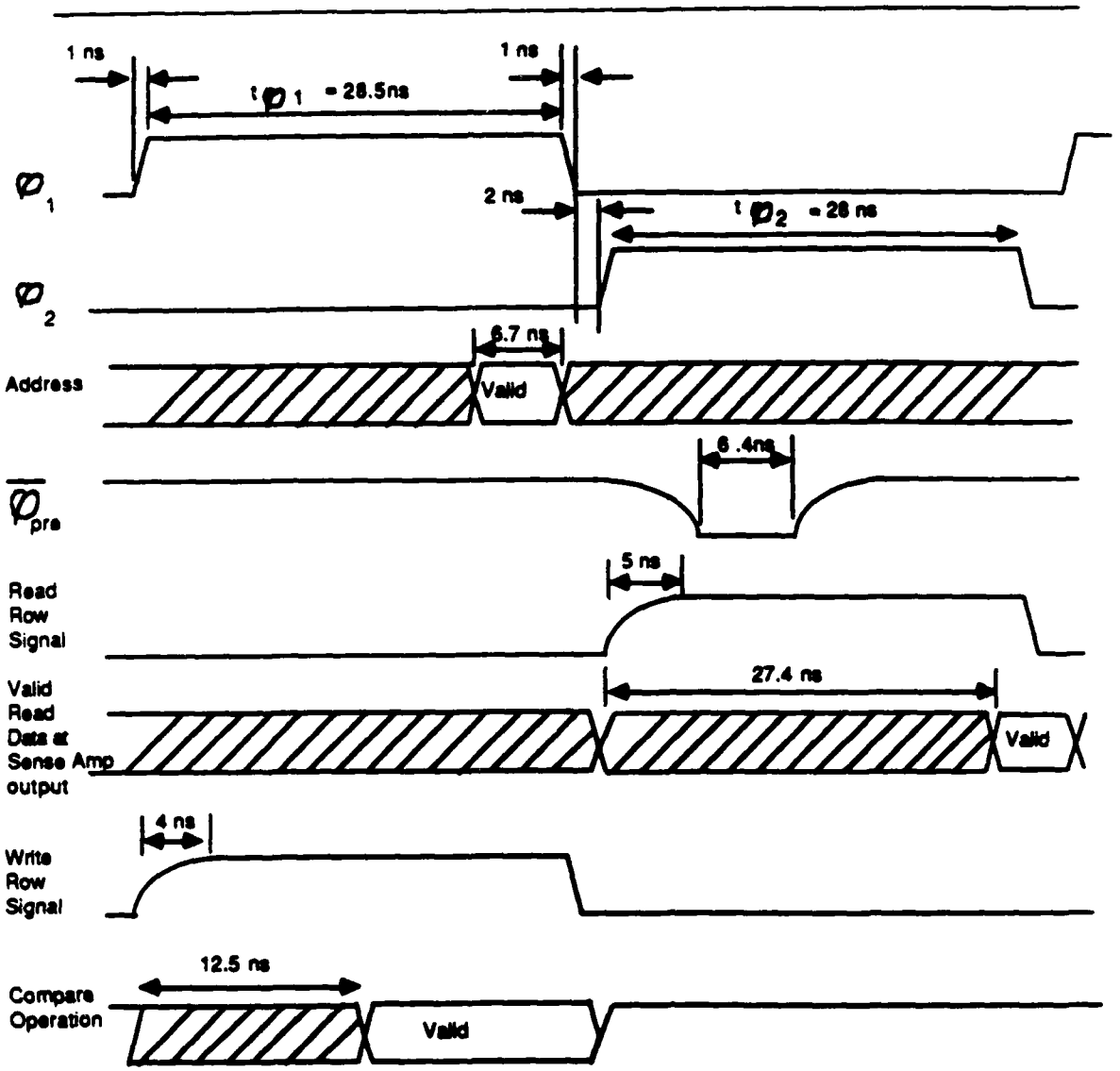


Figure 3.1: Memory Timing

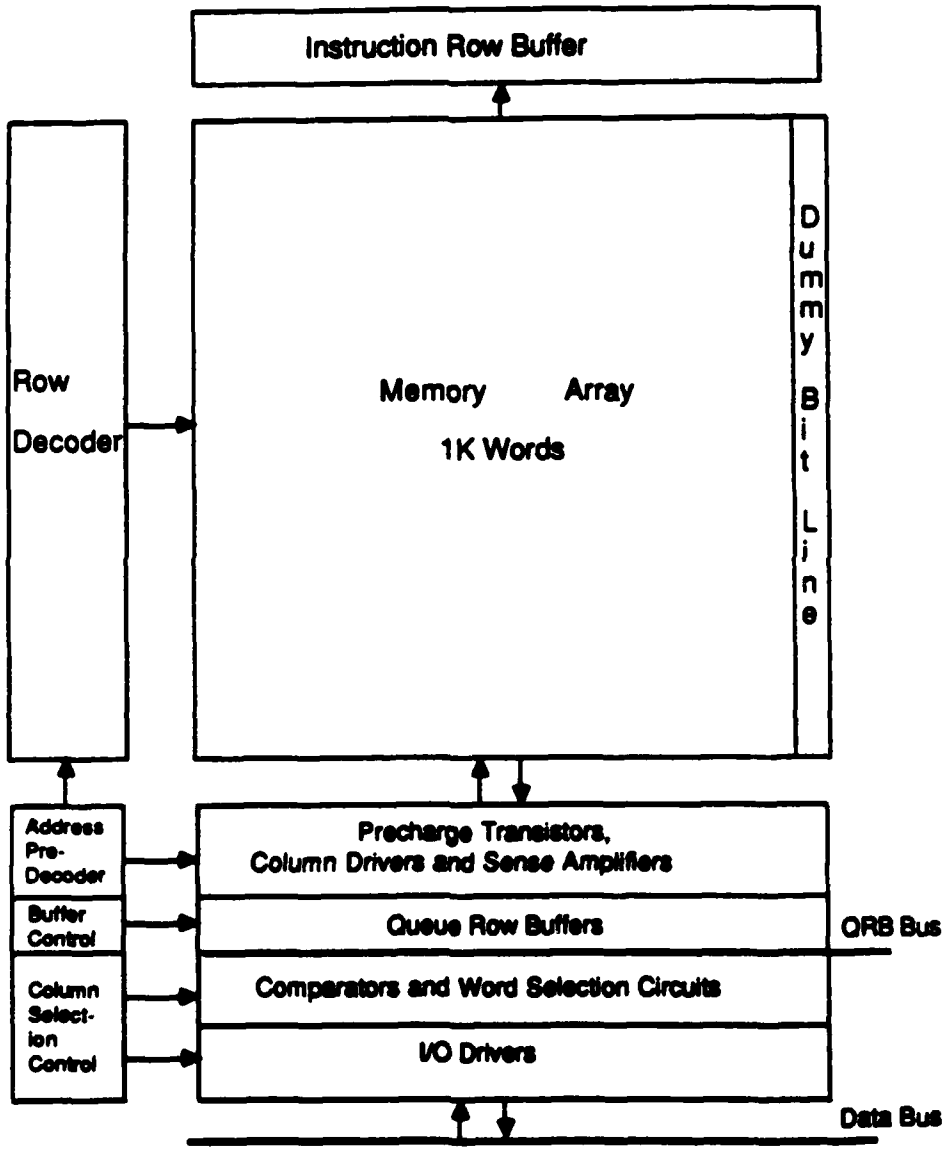


Figure 3.2: Memory Functional Block Diagram

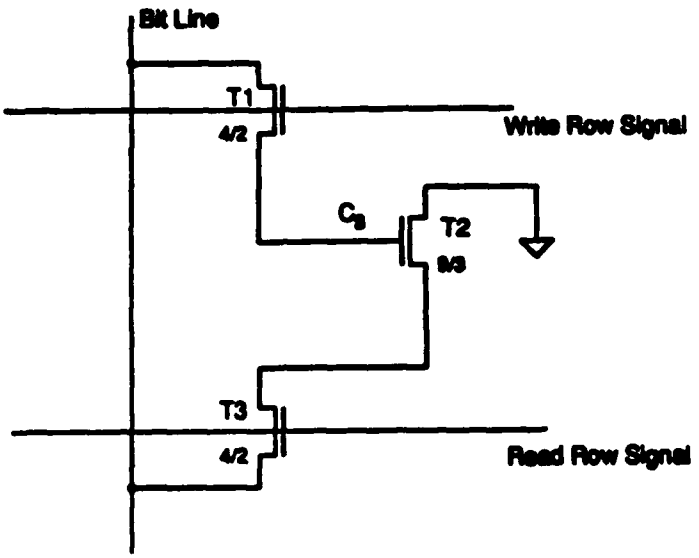


Figure 3.3: Three Transistor Memory Cell

3.2.1 The Memory Cell and Memory array

The memory array consists of 36K three-transistor memory cells. The 3-transistor memory cell is shown in Figure 3.3. Binary information is stored as a charge on a capacitance, C_s. This capacitance is formed from primarily the gate capacitance of transistor T_2, and the diffusion capacitance of the drain of T_1 for a total of $37fF$.

The *row write signal* is high during the write operation. Writing the cell is accomplished by driving the data on the bit lines. The data is passed to the cell through T_1 and stored on C_s. To read the cell, the bit line is precharged high and the *row read signal* is set high. The bit line discharges through T_3 and T_2 only if a logic 1 is stored on C_s, the capacitance of the bit line. The bit line discharges at a rate of 100 mV/ns.

The threshold voltage, V_{tA}, of T_1 limits the stored voltage on C_s. V_{th} is higher than the

threshold voltage of an n-channel transistor with a grounded source due to the back gate (body) effect. An increased voltage difference between the source and substrate increases the back gate factor. SPICE simulations using a back gate factor, γ , of 0.9 $V^{\frac{1}{2}}$ increase V_{th} of T_1 to 2.1 Volts.

The memory cells require refreshing. The stored dynamic charge leaks off due to the subthreshold leakage current of T_1. The drain to source current, I_{ds} ,of T_1 displays an exponential behavior when it is at cut off similar to a reverse biased p-n junction. I_{ds} increases with higher operating temperatures. The frequency of refreshing is a function of the subthreshold current, the capacitances of the cell and the bit line, and the amount of charge allowed to leak without losing the logic value stored in the cell. For example, SPICE simulations allowing the storage voltage to leak to $1Volt$, and operating at 70 deg C requires refreshing every $0.335ms$.

3.2.2 The Address Decoder

The higher order eight bits of the address select a row in memory. Address decoding is performed at two levels. First, each pair of address lines is decoded into one of four address select lines (AS4-AS19). At the second level, one of each four address select lines is input to the row decoder. Figure 3.4 illustrates the two levels of address decoding.

A row decoder is a domino 4-input NAND gate. The row select signal is latched by the falling edge of φ_2. The *row write signal* and the *row read signal* are driven across the array in 4ns and 5ns from the rising edges of φ_1 and φ_2 respectively. Figure 3.5 is a circuit schematic of the row decoder.

3.2.3 The Sense Amplifier

Sense amplifiers speed the detection of a change in voltage on a bit line when it is discharging. The sense amplifier used in the MDP memory is a charge sharing amplifier [DG85]. This

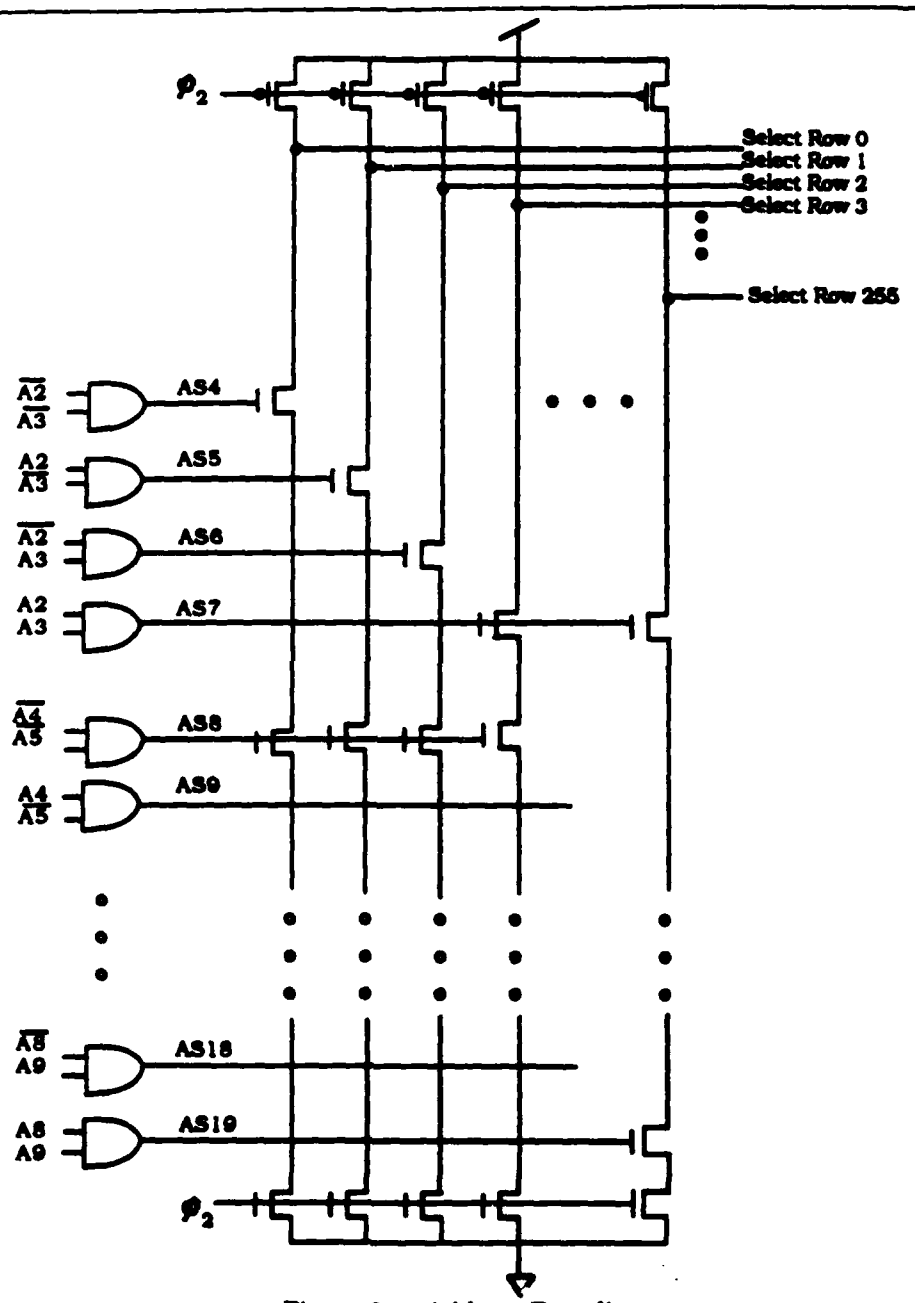


Figure 3.4: Address Decoding

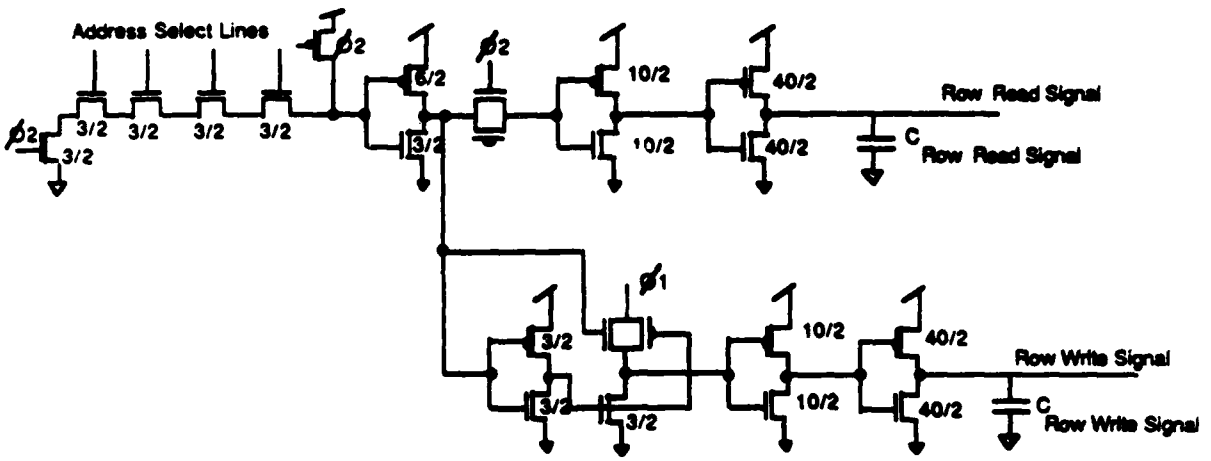


Figure 3.5: Schematic of Row Decoder

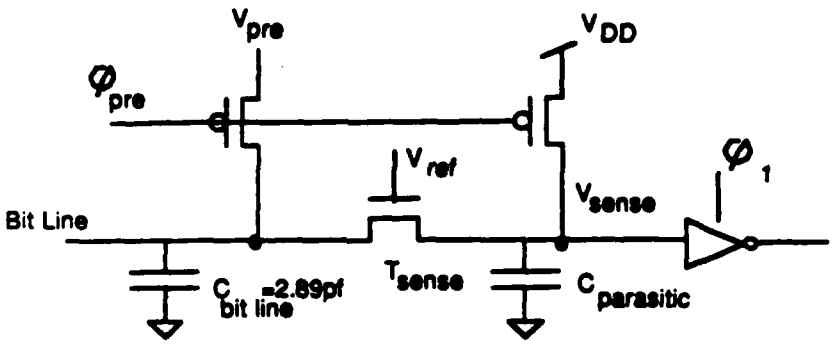


Figure 3.6: The Sense Amplifier Circuit

circuit is illustrated in Figure 3.6. The bit line is precharged to V_{pre}, a few hundred mV above V_{ref}. The other side of the sense amplifier is precharged to V_{DD}. T_{sense} conducts when the bit line voltage drops by the threshold voltage of T_{sense} below V_{ref}. (i.e. $V_{conduct} = V_{ref} - V_{th}$). When T_{sense} is conducting, $C_{bitline}$ appears large compared to C_{sense} and V_{sense} quickly tracks V_b

The high noise margin of this sense amplifier, NM_h, is equal to the difference between the initial precharge voltage on the bit line, V_{pre}, and the voltage at which T_{sense} starts conducting, $V_{conduct}$.

Precharging the bit line to a voltage, Vpre, closer to $V_{conduct}$ reduces the noise margin, NM_h, but allows T_{sense} to conduct faster. This improves the speed at which ΔV_{bit} is sensed. The graph in Figure 3.7 illustrates the trade off between the noise margin and the speed at which T_{sense} conducting for two different values of V_{pre}.

In addition to speeding the sense amplifier, precharging the bit lines to a lower voltage

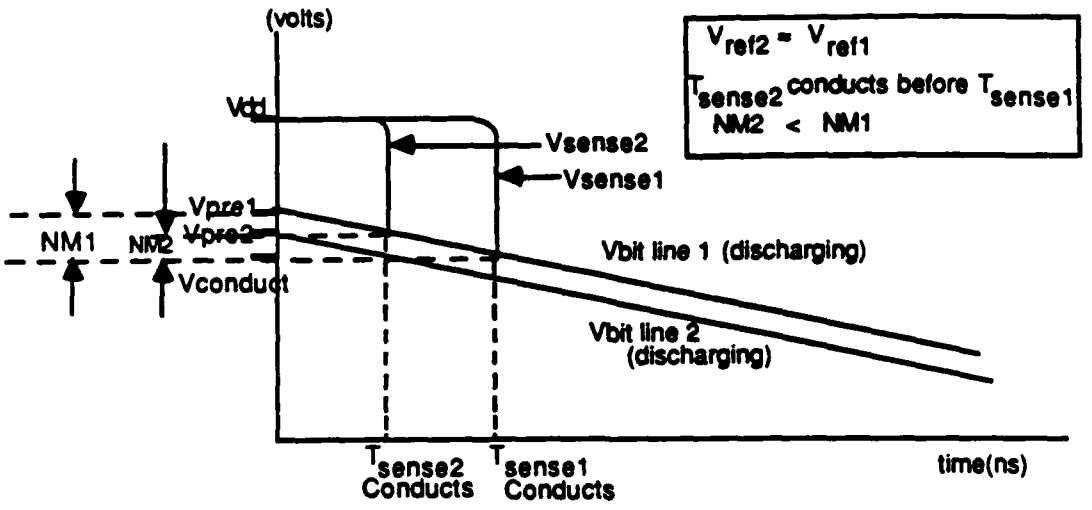


Figure 3.7: Precharge Voltage, V_{pre}, vs. High Noise Margin, NM_h

reduces power dissipation of the array by a factor proportional to V^2.

The inverter connected to the sense amplifier has a trip point equal to $V_{conduct}$. It also corrects the logical value of the data read from the memory array before writing it back into memory.

A sense amplifier for the MDP memory was designed with a noise margin of $500mV$. V_{ref} is set to 3.5 Volts and the bit lines are precharged to 3 volts. The read operation is performed in 9.25ns.

3.2.4 Precharge Circuit

A dummy bit line is used to generate the precharge clock. Figure 3.8 illustrates the circuit and the waveform of $\overline{\varphi_{pre}}$. The dummy bit line is driven low during φ_1. $\overline{\varphi_{pre}}$ goes low with

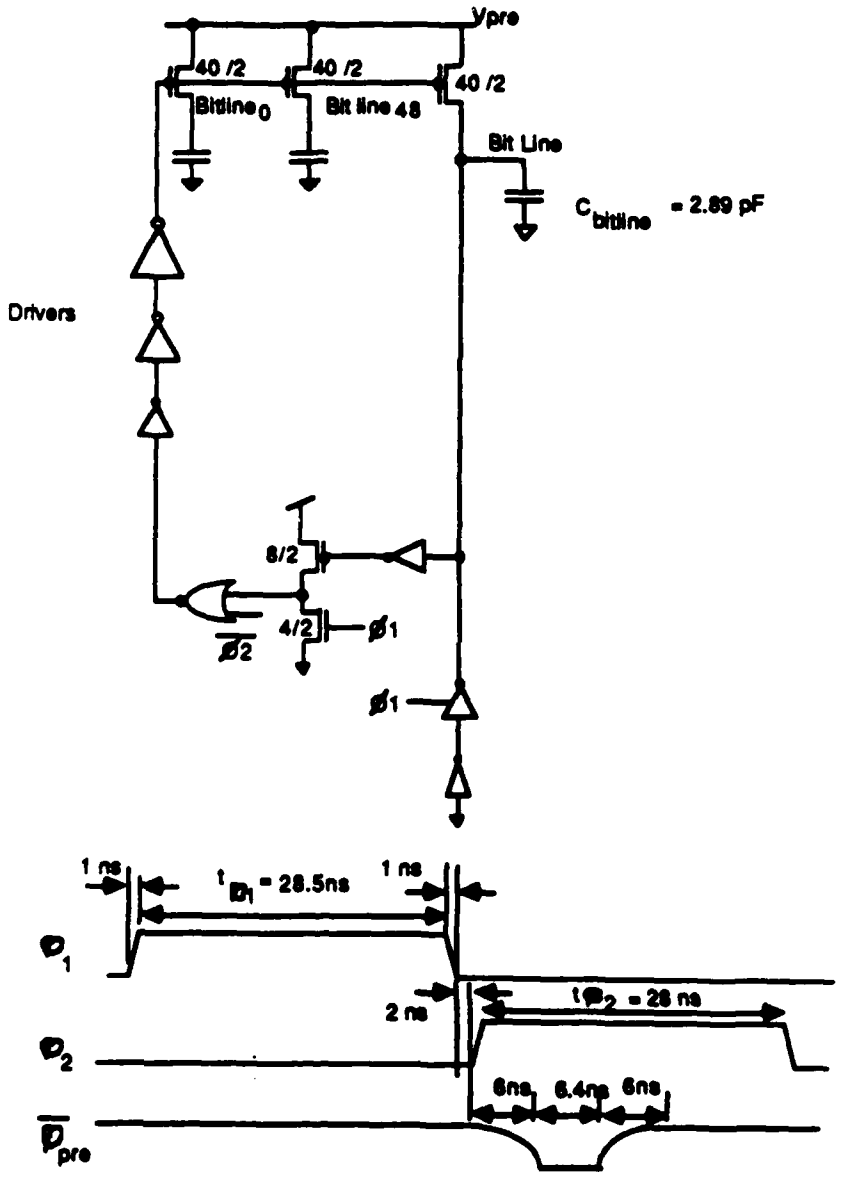


Figure 3.8: The Precharge Clock Generation Circuit and φ_{pre} waveform

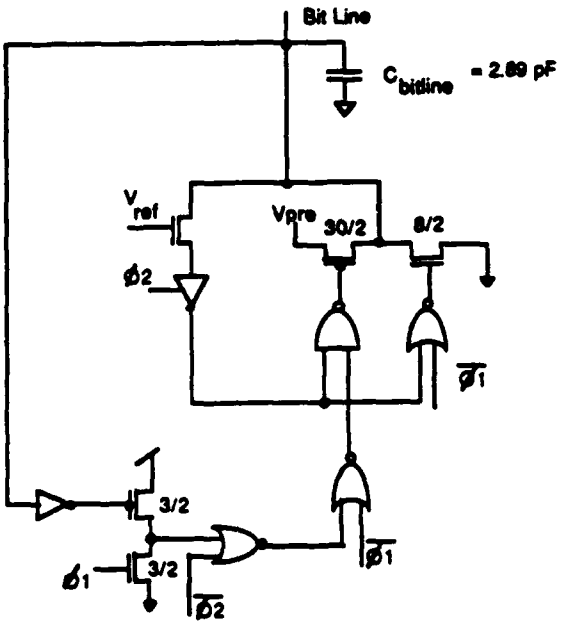


Figure 3.9: Precharge Design Alternative Circuit

the rising edge of φ_2. The bit lines are precharged to V_{pre} through 40μm wide p-channel transistors. The effect of the rising dummy bit line is propagated to set $\overline{\varphi_{pre}}$ high.

Design Alternative :

The precharge operation described above is completed in 18.4ns. Two thirds of the precharge operation is the delay in distributing $\overline{\varphi_{pre}}$ to the precharge transistors. An alternative design is to use the p-channel of the column drivers as precharge transistors. This circuit is shown in Figure 3.9. Optimizing the circuit and increasing the width of the p-channel pull-up circuit from the original 16μ m to 30μm, would allow charging the bit line in 7.5ns. The delay through the feedback path to turn the precharge off is 8ns. This increases the operating frequency to 16.67MHz. Other advantages include the decrease in layout area especially of the precharge transistor's drivers.

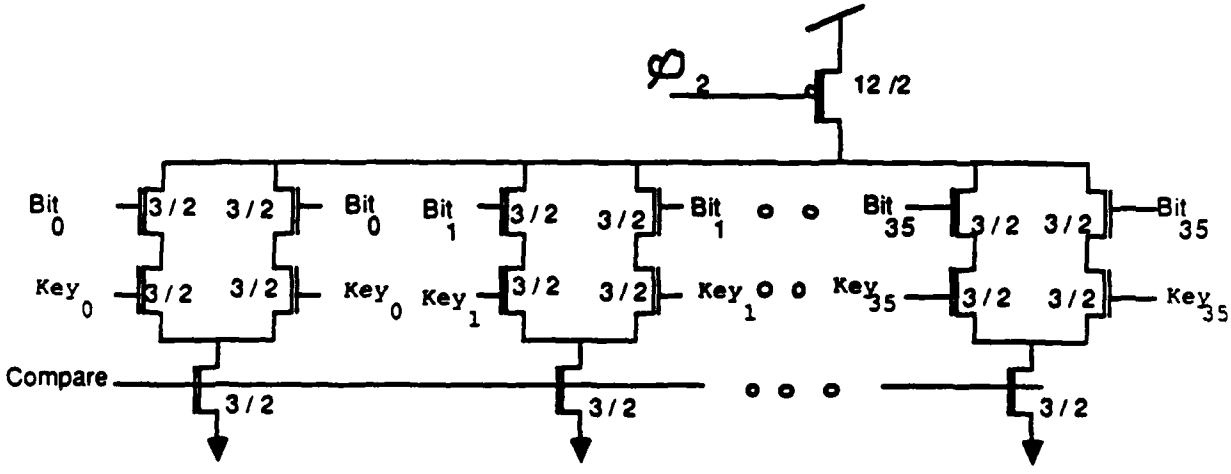


Figure 3.10: The Comparators

3.2.5 Power Dissipation

Precharging the bit lines dissipates most of the power necessary to operate the memory. Charging a bit line to 3 Volts at 15.5MHz, dissipates $0.40mW/bit\ line$. Worst case dynamic power dissipation occurs when all the bit lines are precharged high for a total of 0.6 W.

3.2.6 Comparator

The comparator is used to compare the even words in the selected row with a key when performing a set-associative operation. The circuit used is precharged XOR circuit shown in Figure 3.10

Each bit in an even word has a bit comparator. The 36 bits in each even word share the precharge transistor. The result of each word comparison, the hit_i line, is discharged if

any bit in the key mismatch the prospective bit in that word.

The worst time delay occurs when only one bit in the key mismatchs the prospective bit in an even word. In that case, the discharge time of the *hit* lines is 5.5ns.

3.2.7 Column Selection

While row selection generates the y-address of the memory array, column selection generates the x-address. Column selection is a function of the two low order address bits, the *hit* lines, and the operation being executed. Selecting a group of columns specifies a word, a pair of words, or a row to be operated on. The column circuitry in the memory is presented in detail in Appendix B.

3.2.8 The Row Buffers

Each row buffer consist of 144-cell register. The Instruction Row Buffer (IRB) is placed at the top of the memory array to facilitate its access. A replica of the sense amplifier and a clocked inverter, is used to read the data into the IRB. Data routed to the Queue Row Buffers (QRB) are multiplexed on a 36-bit data bus. The data in the buffer is driven on the bit lines through clocked inverters.

3.3 Layout

The memory cells are arranged in 256 rows and 144 columns. The memory array and its peripheral circuitry occupy 16.5 Mλ^2 and 7.2 Mλ^2. A p^+ guard ring surrounds the array to reduce the injection of minority carriers into the P-well.

Bit lines and ground lines run vertically in parallel in the first metal layer. *Row read lines* and *row write lines* are routed horizontally in polysilicon. They are strapped to the second metal layer to reduce their resistance.

To reduce the area of each cell, horizontally adjacent cells share the contact to the ground line. Vertically adjacent cells share the contact to the bit line. Fig 3.11 illustrates the layout of a 4 adjacent cells. To compact the layout of the peripheral circuitry, the bit lines are interleaved so that bit_i of each word in the row are grouped together.

The layout of the peripheral circuitry was challenging. To maximize the density of the layout, the circuitry for address decoding and row signals had to pitch match vertically with the memory cells. The column selection circuitry had to pitch match with the horizontal pitch of the memory cell.

3.4 Summary

This chapter covers the design of prototype MDP memory. The memory is 256 rows by 144 columns. Each memory cell is a 3-transistor DRAM cell. Peripheral circuitry allow accessing the memory by index or as a set associative cache. The memory cycle is 64 ns.

The logic design was checked using a logic simulator, RNL. Timing was verified through the circuit simulator, SPICE. This memory design reports typical speeds that were achieved by averaging the results of simulations using the fast/fast and slow/slow process corners. The slow/slow process corner performs at half the speed of the fast/fast process corner. Layout was done through the layout editor Magic. The 2μm CMOS process electrical parameters used in SPICE simulations are listed in Appendix C. Schematics are found in Appendix B.

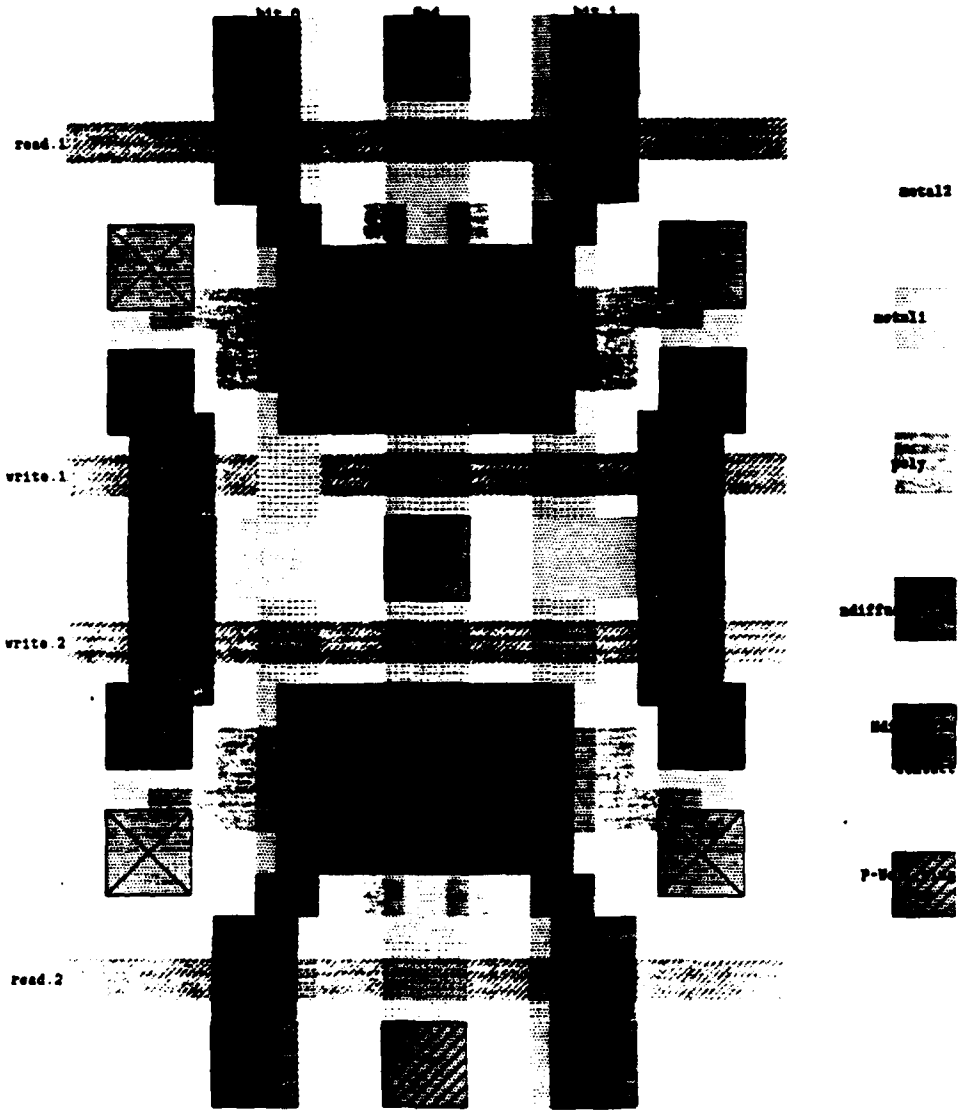


Figure 3.11: Layout of Four Neighboring RAM Cells

Chapter 4

A Prototype Memory

In plucking the fruit of memory, one runs the risk of spoiling its bloom.

— JOSEPH CONRAD, in *The Arrow of Gold, 1919, Author's Note*

A prototype memory chip was fabricated to evaluate the design presented in Chapter 3. To function correctly, every memory cell should be able to store a binary value. The decoder circuit should access every memory cell when correctly addressed. Storage nodes should hold charge until the next refreshing cycle.

This chapter describes potential problems in the prototype memory and the implementation of on-chip test circuitry. The last section includes the results of testing the prototype memory.

4.1 Potential Problems

This section describes some potential problems with RAM structures. Other problems include faulty address decoding and column selection that prohibit accessing the desired data.

4.1.1 Capacitive Coupling

Capacitive coupling in the memory array could cause altering of the stored data or increasing the need for a refresh cycle. Capacitive coupling includes:

1. Parasitic Coupling:

   The small cell area increases the effects of parasitic coupling. For example, when reading a memory cell, the read transistor, T_3 in Figure 4.1, is conducting and the bit line is precharged to V_{pre}. Charge is shared between $C_{bitline}$ and the capacitances of the source of T_3, C_{s3}, and the drain of T_2, C_{gd2}. The gate capacitance of T_2, C_{gb2}, has a lower value when it is non-conducting due to the decreased number of inversion layer electrons. The coupling between drain of T_2 and its gate could yank this voltage high enough to turn on T_2. The sense amplifier would sense the small discharge in the bit lines and read the data as a high. To minimize this parasitic coupling, the perimeter of C_{gd2} was minimized in the layout.

2. Inter-bit line Coupling:

   Faulty reading or writing of memory cells could occur because of interaction between cells that share signal lines or bit lines. Usually such interactions are caused by repeated patterns.

   For example, writing a 0 in a memory cell, and writing 1s repeatedly in other cells in the same column could cause an increase in the subthreshold current. Coupling between a deselected row write signal and a high-driven bit line could yank the voltage on the gate of T_1 of the memory cell. The increase in subthreshold leakage current demands more frequent refreshing. If the yanked voltage reaches above the threshold voltage of T_1, data in a deselected row could be modified. The write row signals are always driven (i.e. not floating) to minimize the effects of coupling.

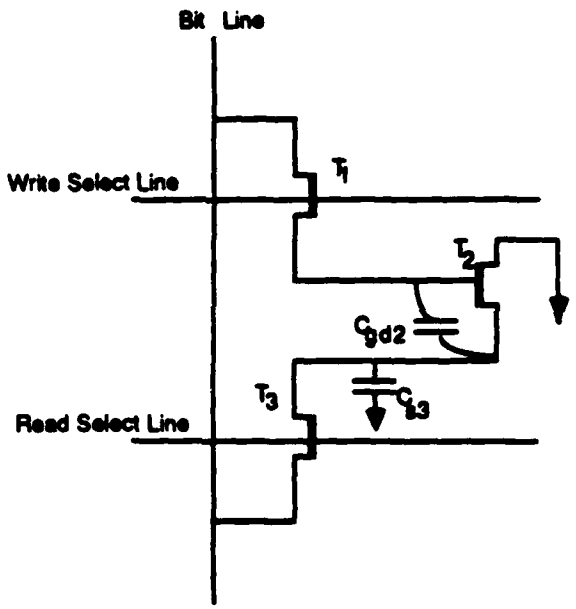


Figure 4.1: Parasitic Coupling

4.1.2 Soft Errors

A soft error is the change in the stored data or logic value. Soft errors in VLSI structures are mainly caused by alpha particles. Alpha particles are doubly ionized He atoms emitted during the radioactive decay of uranium or thorium which is found in VLSI packaging material. Soft errors are especially present in DRAMs because of the dynamic charge stored in each cell and the high packing density of the DRAM structure.

As an alpha-particle hits an active device and travels through the material, electron-hole pairs are generated. N-regions in the memory cell collect electrons. The major collection mechanisms are drift and field-funneling [Bre88]. Drift is the movement of carriers due to an electric field. Field-funneling is the modification of the electric field due to drift. The soft error rate is a function of the collection efficiency and the memory cell's storage area. If the charge collected exceeds the critical charge to store a logic 1, Vh, a soft error will occur.

When an alpha particle hits the drain or gate of T_1, a track of electrons and holes is generated. The electrons are carried into the drain by the horizontal electric field. They neutralize the positive charges stored on the storage capacitance. If the electrons collected by the drain of T_1 causes the stored voltage to drop below the threshold voltage of T_2, a soft error will occur.

A typical alpha particle of $3.6MeV$ generates 1.4 x 10^6 electron-hole pairs. The critical voltage that would cause a soft error is 2.0 volts. The critical charge is 0.462 x 10^6 holes. An alpha particle hit on the drain or gate of T_1 would cause a soft error. Assuming an alpha particle flux rate of 0.1 α / $cm^2.h$, and that on average half the alpha particle hits cause a soft error, we expect an error rate of 5.4 x $10e^{-3}$ errors/h. Error detection and correction circuitry would eliminate those errors.

4.2 Test Circuits

The purpose of the on-chip test circuitry is to trace internal wave forms and to facilitate generating test vectors to test for problems explained in Section 4.1. All the circuit schematics of the test circuitry are illustrated in Appendix D.

4.2.1 Voltage Comparators

Five clocked voltage comparators were placed on the memory test chip. The purpose of the comparators is to provide an on-chip sampling scope to observe important internal waveforms. Outputs of the voltage comparators were brought off-chip. The comparator was designed to detect a difference of 110 mV between its input voltages. A bit line, the dummy bit line, the precharge clock, the row signals were inputs to the comparators.

The advantage of using comparators is to obtain an accurate measure of the internal wave forms. The capacitance seen by the comparator's output causes a delay in the result of the comparison. It does not distort the original signal. Probing and routing the desired signal to output pads distort the signals by loading them with undesired capacitances.

4.2.2 RAM Test Patterns

Goals of Test Patterns

We have chosen three different test patterns that check for different possible malfunctions in the memory array. These patterns include a Checkerboard pattern, the Walking 1's and 0's pattern, and a random pattern [BF76].

The checkerboard pattern tests for possible interaction between adjacent rows and columns of the array. One logic value is written in all the even cells in a row, and the complementary logic value is written in the odd cells. All memory locations are verified for

the proper value. This test is repeated for the two complementary patterns.

The Walking 1's and 0's pattern checks that every memory cell can be set to both logic values without influencing any other adjacent cell. It also checks for correct address decoding. The test starts be setting every memory cell to 0. One memory cell is altered at a time. After every alteration, the whole memory array is read. The test is repeated for complementary logic values.

Although the two patterns above could be used to measure the memory refresh time, a random pattern is implemented to measure this refresh time. A pseudo-random pattern is written into memory. The memory is still for the expected refresh period. The memory is read check for changed values.

Implementation of Test Patterns

An on-chip circuit was designed to generate the different test patterns and addresses at which data is to be written. Data comparators are used to compare a certain test pattern with data read form memory. An address register and comparator hold and compare a certain address with the current address. The control pins for these circuits are routed to input pins. The results of the address and data comparators are routed to output pins. An off-chip ROM uses these pins to generate the correct test sequence. Off-chip sequencing eases sequencing the test patterns and allows refreshing when needed.

4.2.3 The Precharge Circuit

In case of the failure of the self-timed precharge circuitry, an off-chip precharge clock was provided. A select pin allows the bit lines to precharge using this off-chip clock or the on-chip generated precharge clock. The precharge phase occurs before the read phase, φ_2, as shown in Figure 4.2.

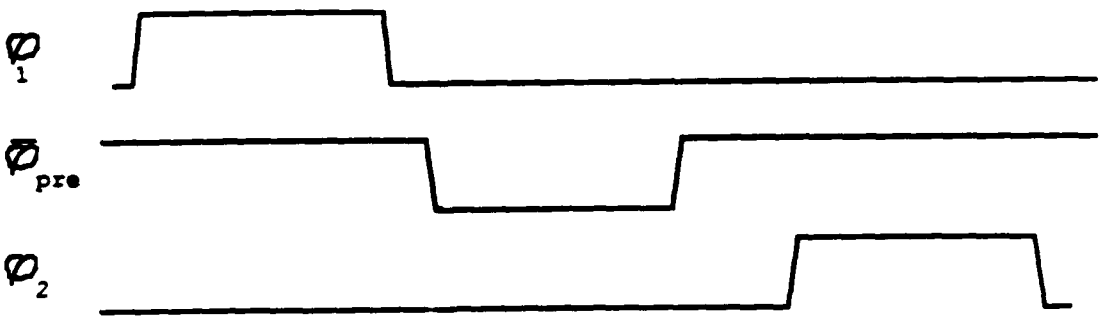


Figure 4.2: Three Phase Test Clock

4.3 The Test Chip

A prototype memory was fabricated using a 2μ double-metal CMOS MOSIS process. It was packaged in an 84 pin package. A picture of the chip and it's floor plan are shown in Figures 4.3 and 4.4. A listing of the pinout is provided in Figure 4.5.

4.4 Testing the Prototype

A test fixture was built for the prototype memory chip. A Digital System Analyzer (DAS) generated the clocks and the control signals and collected digital data. An oscilloscope was used to observe output waveforms.

The bidirectional Data pins (pins 45-62, and pins 65-82) prohibited generating and acquiring data using the DAS' pods. Therefore, the DAS data vectors were written through buffers (6 RCA CD74HCT245Es) to the memory chip when writing data. Data was acquired at the chip's data pins when reading data.

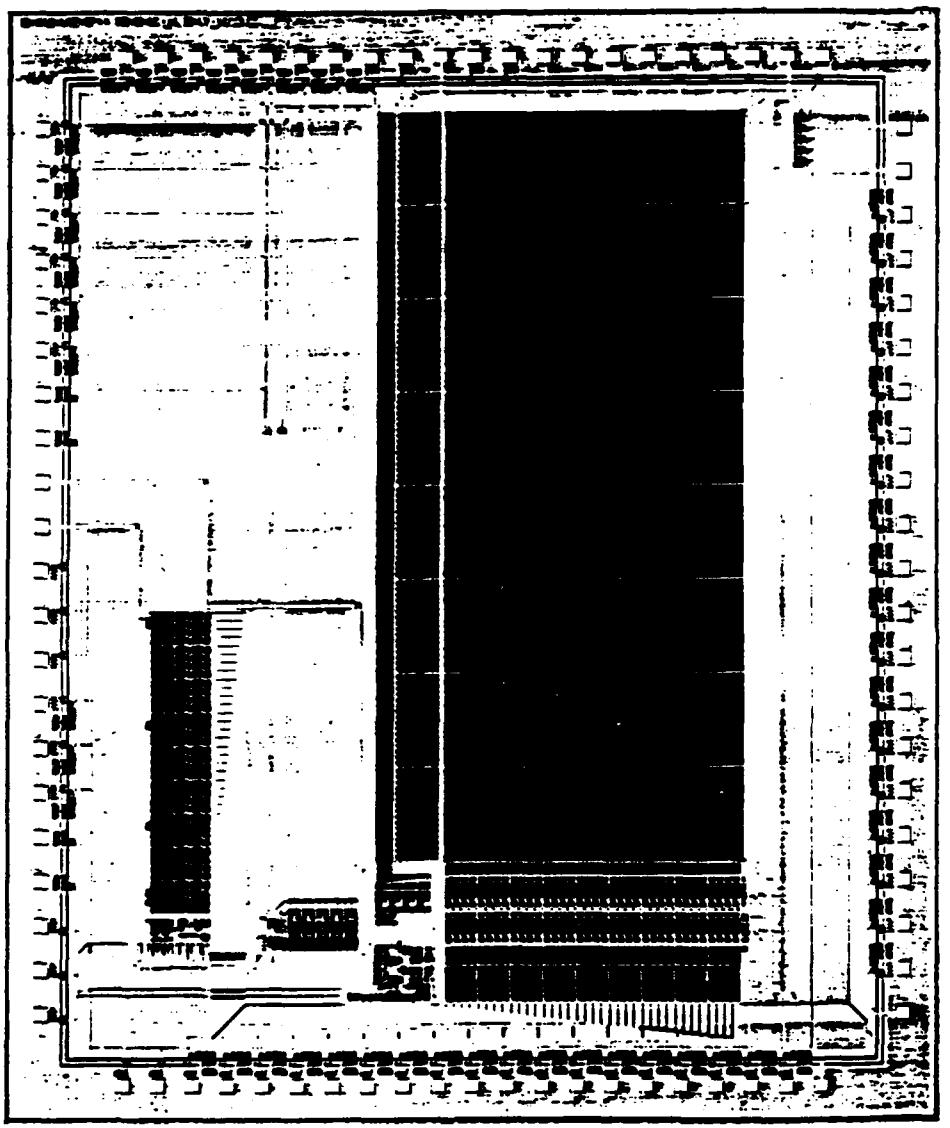


Figure 4.3: The Memory Test Chip Picture

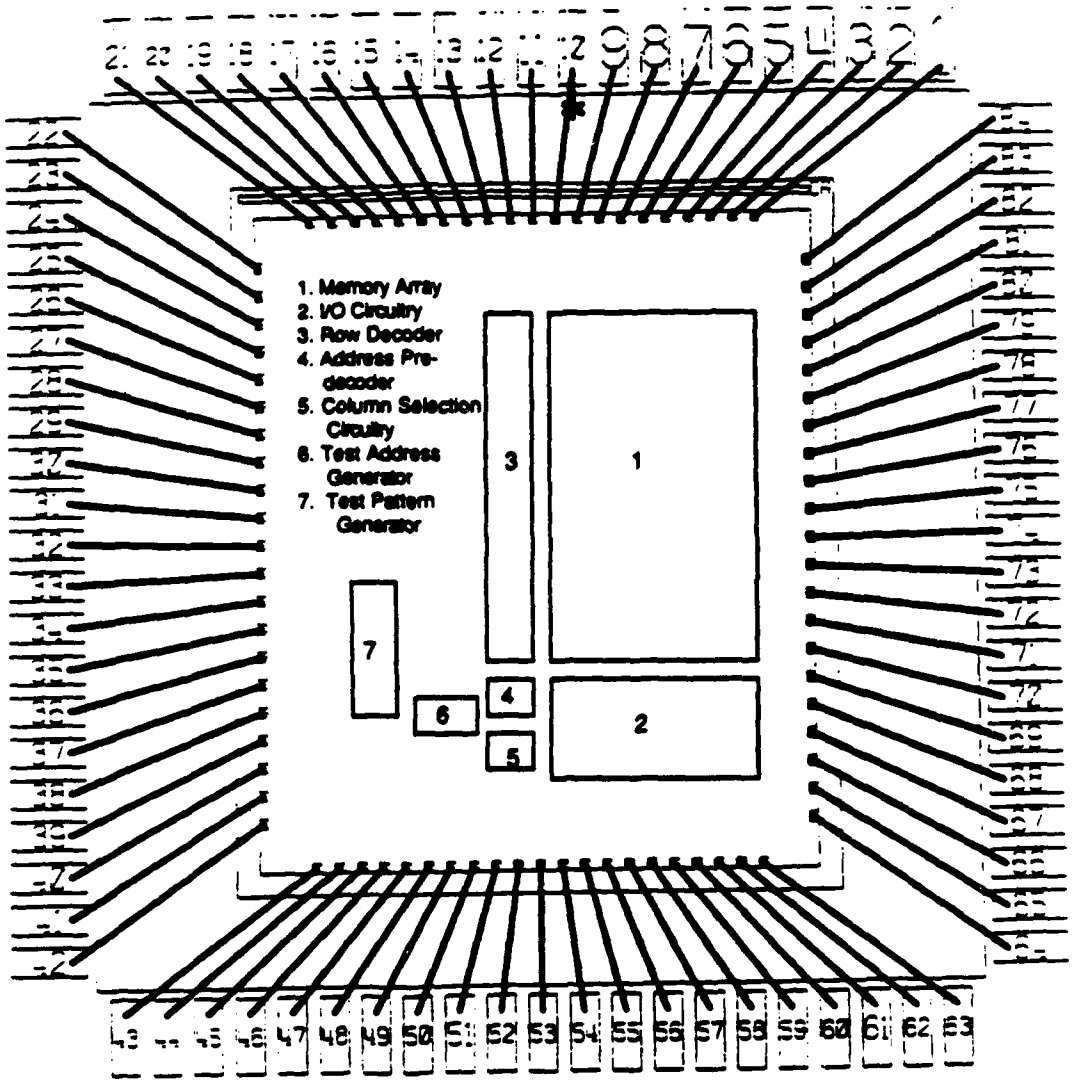


Figure 4.4: Test Chip Floor Plan

PIN	Description
1	Voltage Comparator Sampling Clock
2	VDD
3	[illegible]1
4	[illegible]1
5	[illegible]2
6	[illegible]2
7	[illegible] pre
8	Vpre
9	Sense Amplifier Reference Voltage
10	Substrate Bias
11	Switch-on/off-chip precharge
12	GND
13	Switch-on/off-chip Address Generation
● 14	Reset Address Counter. / Write Row Signal
● 15	Increment Address Counter / Read Row Signal
● 16	Enable into Address Register / Bit line
● 17	Compare Address / Dummy bit line
18	Add.9
19	Add.8
20	Add.7
21	Add.6
22	Add.5
23	Add.4
24	Add.3
25	Add.2
26	Add.1
27	Add.0
28	Result of Address Compare
29	Cout of Address Counter
30	GND
31	VDD
32	Sel_A
33	Sel_B
34	Sel_C
● 35	Reset test Pattern / Precharge Clock
36	Shift LSR / Hit0
37	Shift RPG / Hit1
38	Result of Data Compare
39	Count of LSR
40	reset random
41	Sel_1
42	Sel_2
43	Sel_3
44	En Data pads as outputs
45	Data.36
46	Data.35
47	Data.34
48	Data.33
49	Data.32
50	Data.31
51	Data.30
52	Data.29
53	Data.28
54	Data.27
55	Data.26
56	Data.25
57	Data.24
58	Data.23
59	Data.22
60	Data.21
61	Data.20
62	Data.19
63	GND
64	VDD
65	Data.18
66	Data.17
67	Data.16
68	Data.15
69	Data.14
70	Data.13
71	Data.12
72	Data.11
73	Data.10
74	Data.9
75	Data.8
76	Data.7
77	Data.6
78	Data.5
79	Data.4
80	Data.3
81	Data.2
82	Data.1
83	GND
84	Reference Voltage for Voltage Comparator

● (Outputs of voltage comparators)

Figure 4.5: Test Chip Pinout

Testing the chip was partially successful. The waveforms of the *read row signal* and the *write row signal* were observed through the voltage comparator's outputs. Figure 4.6 has pictures of φ_2, and the *read row signal* when it is at 1 and 4 Volts. Figure 4.7 is a picture of φ_1, and the *write row signal* when it is at 1 and 4 Volts. The measured delay between the row signals and the clock edges (29.62 ns and 31.60 ns) were slower than SPICE simulation results. This is mainly due to the differences in T_{OX}, the oxide thickness, between the SPICE deck used for simulation (T_{OX} ranged between 22.5 and 27.5 nm) and the MOSIS parametric test results (T_{OX} is 40.6 nm). Figure 4.8 has multiple-exposures of the *read row signal* with V_m of the voltage comparator set at different voltages. From measurements and from these photgrahps it is evident that the row signals have a very short rise time. The address and data comparators used in the test circuits operate correctly.

An unplugged P-Well in the column select control circuitry was fatal to writing and reading data from the memory array. We were unable to observe the bit lines due to a 4λ opening in the routing of the output of the bit line voltage comparator. In the test circuitry, a design rule violation that was not detected by the layout design rule checker caused a failure in the random pattern generator. A misconnection in the Walking '0 and '1 generator produces a wrong pattern.

4.5 Summary

This chapter describes potential problems in the memory due to coupling capacitances and soft errors. Voltage comparators and pattern generators and comparators were placed on-chip to test for errors. A prototype memory was fabricated and tested. An error (an unplugged P-Well) in the control circuitry prevented accessing the memory array. The row decoder and other parts of the test circuitry were functional. A corrected version of the memory prototype will be sent for fabrication within the next two weeks.

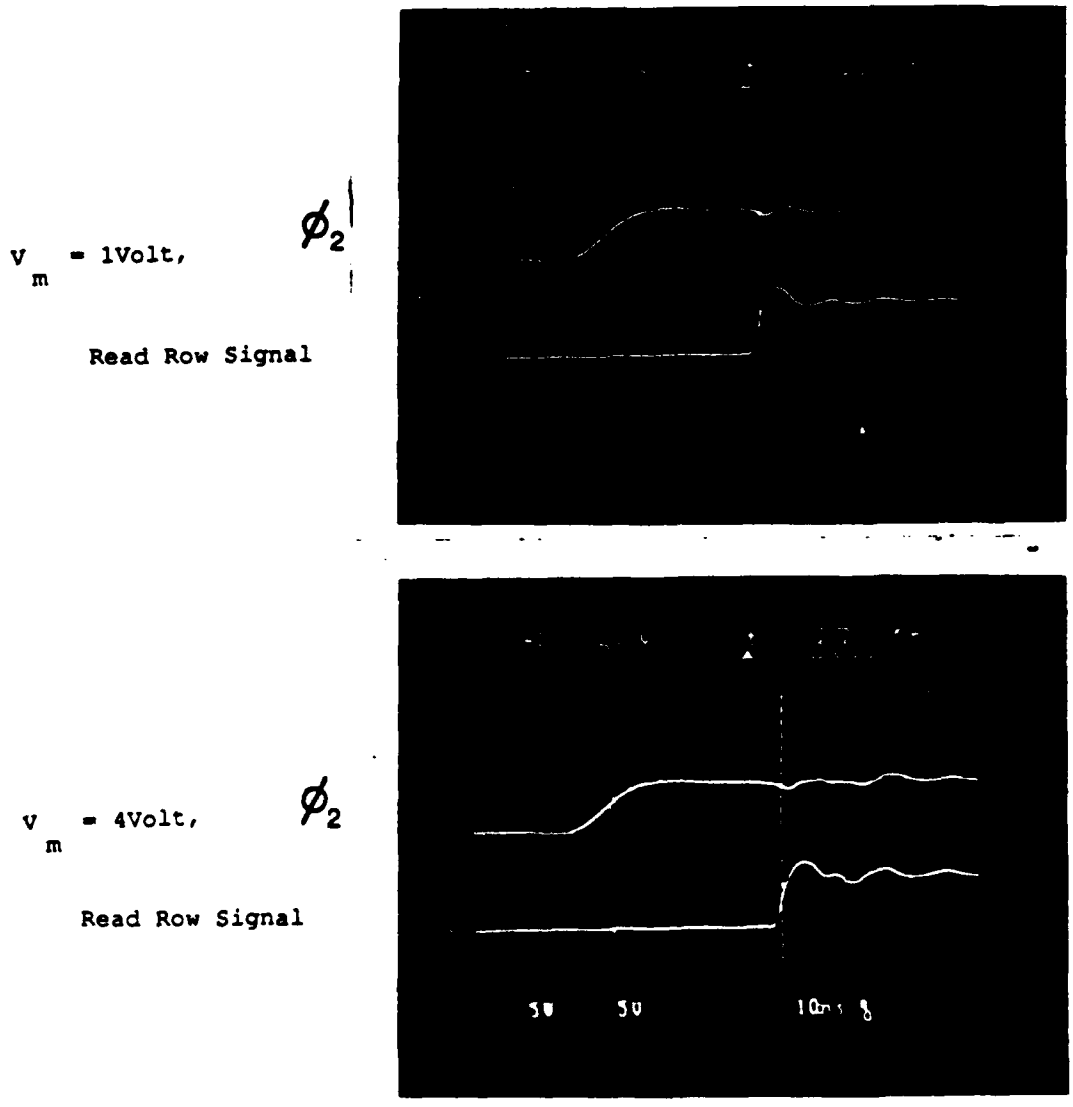
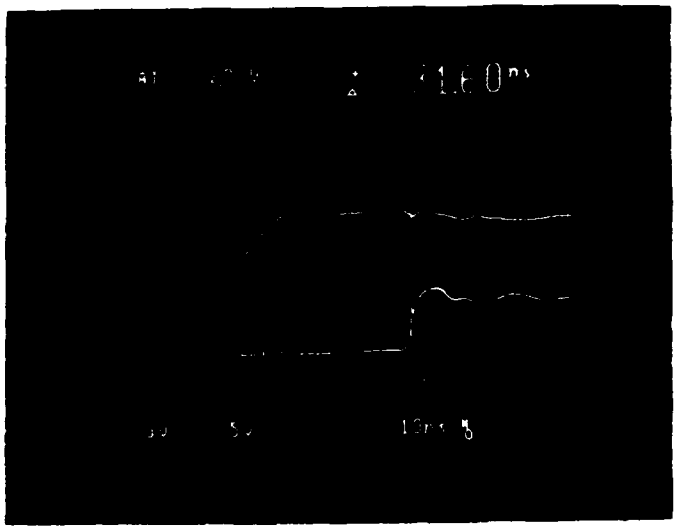


Figure 4.6: φ_2 and *Read Row Signal* Waveforms

V_m = 1Volt, ϕ_1

Write Row Signal



V_m = 4Volt , ϕ_1

Write Row Signal

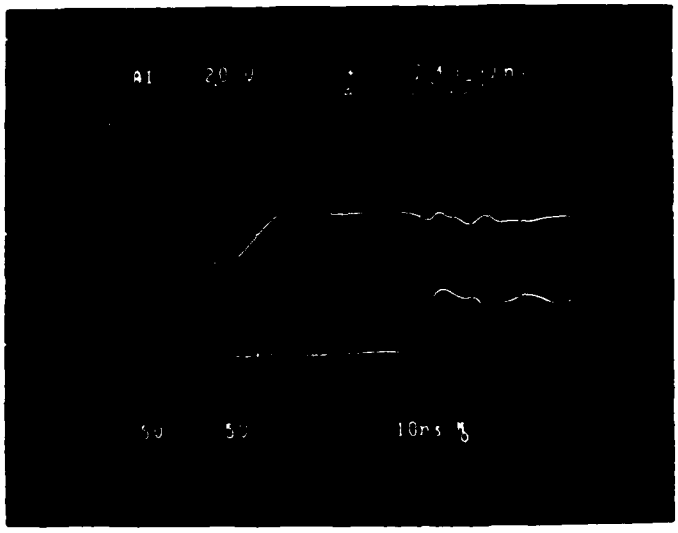
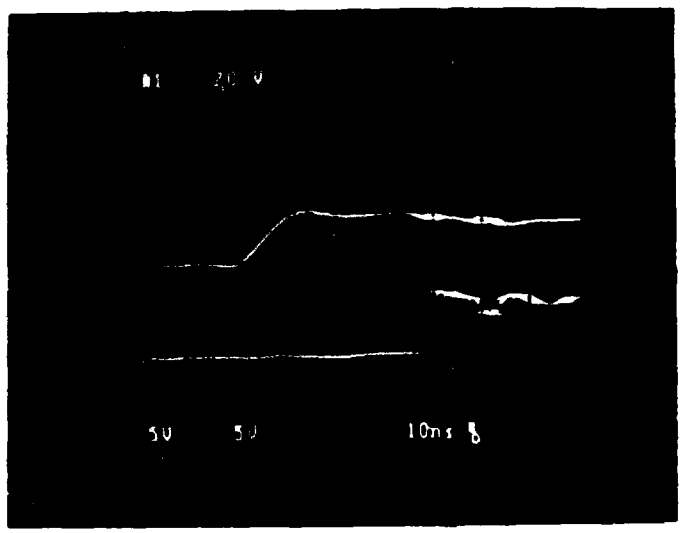


Figure 4.7: φ_1 and *Write Row Signal* Waveforms

V_m = 1,2,3,4 Volt, ϕ_2

Read Row Signal



V_m = 3, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5 4.75, 4.9 Volt, ϕ_2

Read Row Signal

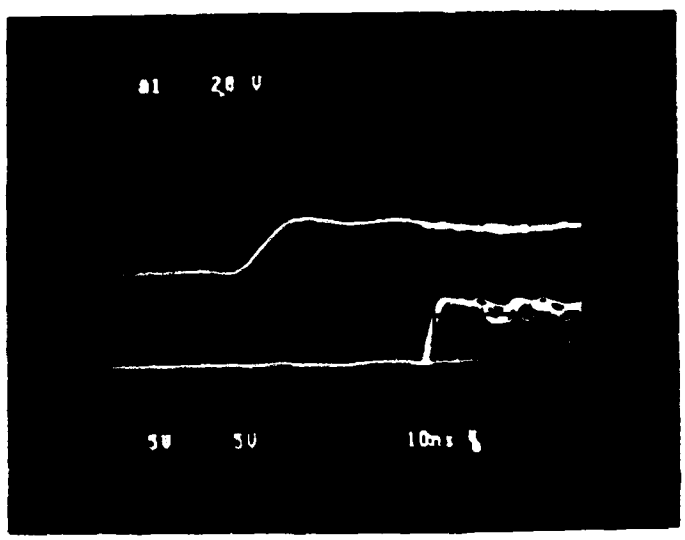


Figure 4.8: Multiple-exposure of *Write Row Signal* with Different V_m

Chapter 5

Evaluation of Architectural Features

Four for the price of one!

— STORE ADVERTISEMENT

The MDP memory microarchitecture introduces a concept of *row operations*. A row (four words) of memory is fetched simultaneously through special row buffers. Row fetching requires only one memory reference. Fetching four words sequentially requires four references.

Each buffer costs an area 0.7 Mλ^2 the complexity of managing it. This chapter evaluates the performance of the Instruction Row Buffer and the Queue Row Buffer.

5.1 The Instruction Row Buffer

A row of memory is loaded into the Instruction Row Buffer (IRB) while executing the instruction stream. This operation increases the memory bandwidth by making more memory cycles available to the executing program.

Two factors influence the performance of the IRB: the basic block size and it's alignment in a memory row. A basic block is a section of contiguous code which does not branch when executed. Both branching and short code require flushing the IRB frequently. Therefore, better performance is achieved as the basic block size increases. The beginning of a block is aligned in any slot in the row with equal probability. The effects of the alignment are more noticeable for smaller block sizes.

The following equations are used to calculate the number of row fetches, *RFetches*. BCA and WCA refer to best and worst case alignment respectively.

$$RFetches_{BCA} = \begin{cases} \frac{w}{4} & \text{if } w \bmod 4 = 0 \\ \frac{w}{4} + 1 & \text{otherwise} \end{cases}$$

$$RFetches_{WCA} = \begin{cases} RFetches_{BCA} & \text{if } (w-1) \bmod 4 = 0 \\ RFetches_{BCA} + 1 & \text{otherwise} \end{cases}$$

The probability of worst and best case alignment is a function of the number of words in a block.

$$P(BC) = \frac{(w-3) \bmod 4}{4}$$

$$P(WC) = 1 - P(BC)$$

Figure 5.1 illustrates the use of these equations for block sizes 5-8.

The gain in row fetching is the fraction of words that are fetched from the IRB without memory access, i.e.

$$Gain = \frac{w - RFetches}{w}$$

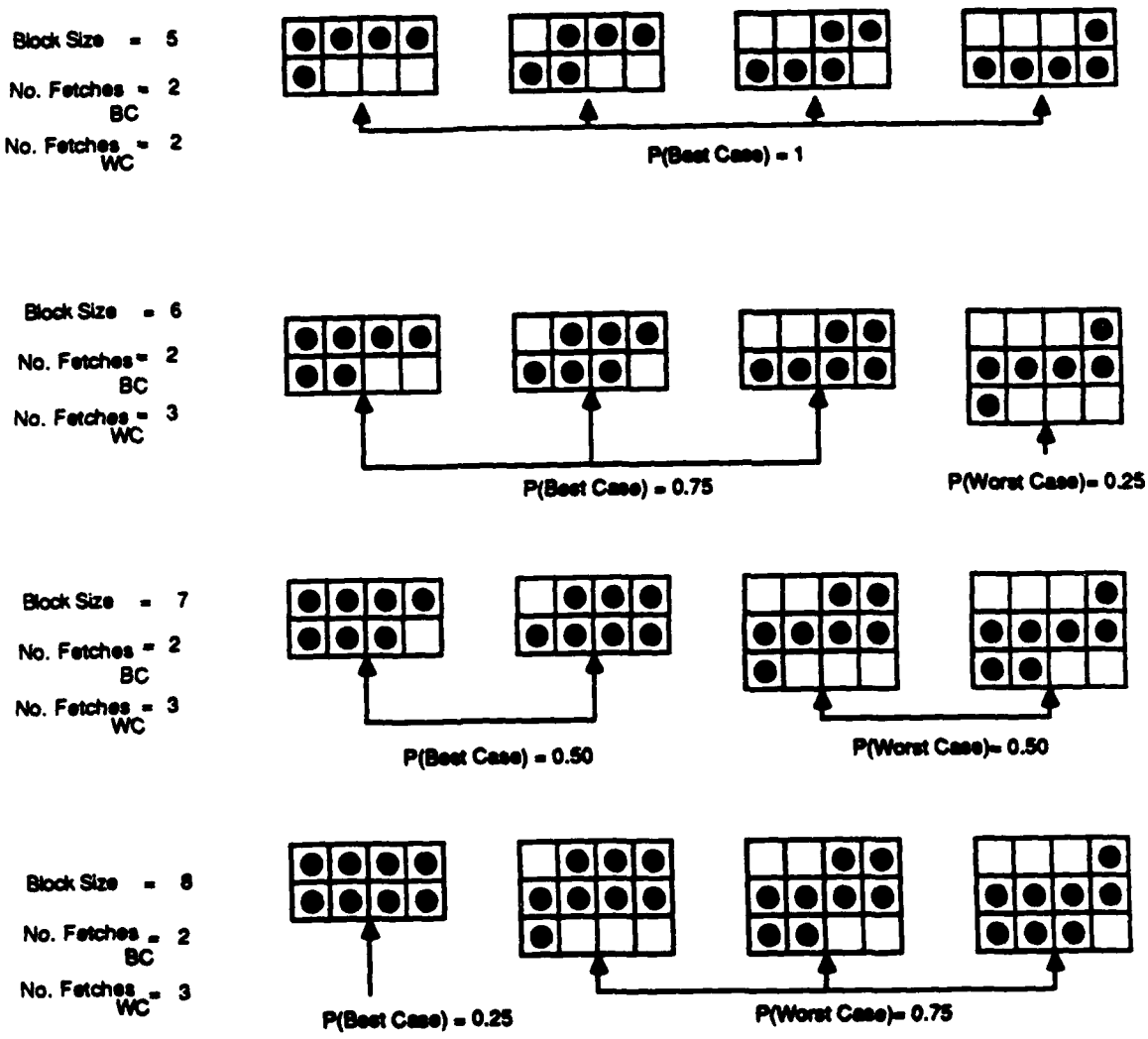


Figure 5.1: Some Block Sizes and Possible Alignments

$$Gain_{average} = P(BC) \times Gain_{BC} + P(WC) \times Gain_{WC}$$

Figure 5.2 is a graph of the gain vs. different block sizes. From this graph, we can conclude:

1. The maximum gain using row fetching is 75%. (i.e. Fetching every four words require at least one memory reference.)
2. For average block sizes (5-10 words) , the minimum gain is 50%. The IRB eliminates at least half the memory references.

5.2 The Queue Row Buffers

Messages arriving from the node's network are enqueued in one of the Queue Row Buffers (QRB). The buffer is written into memory when it becomes full or when the network signals an end of a message. Similar to the idea of the IRB, the QRBs reduce the memory cycles needed to enqueue a message.

The performance of the QRBs is a function of the number of words arriving from the network per cycle. Once a message's first word arrive, it is very likely that the remainder of the message follows at a rate of 1 word/ cycle. The bidirectional nature of the communication channels [DS87] could cause a slower arrival rate of 0.5 word/ cycle. Therefore, decreasing memory accesses to write new messages permits the execution of more memory instructions if contained in the executing program.

When the processor is idle, enqueueing a message via the QRBs delays execution by 4 cycles. Simulation results [Son88] indicate that the message arrival rate at a node is 0.0014 messages/ cycle in a 1-K node machine with a network capacity of 45%. i.e. A message (6

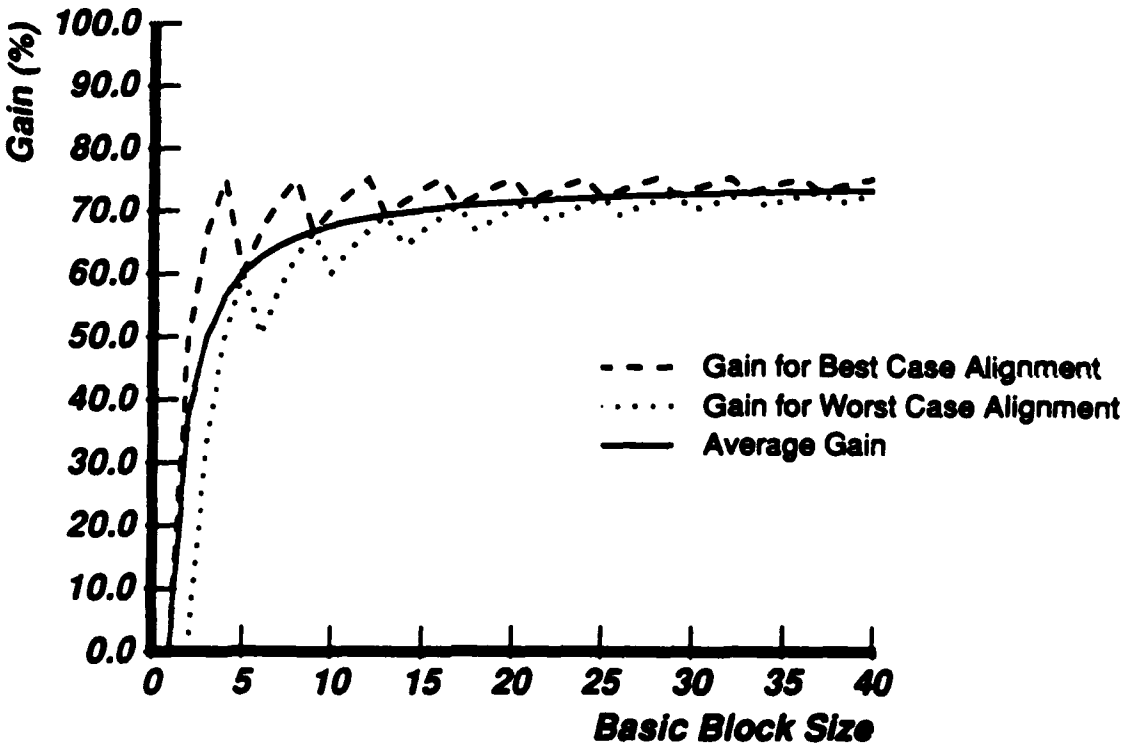


Figure 5.2: Performance Gain of IRB vs. Basic Block Size

words) is expected to arrive at a node every 714 cycles. Since an average program's length ranges from 10-20 instructions, it is likely that messages arrive at idle processors.

5.3 Summary

The purpose of *row operations* is to reduce the number of memory cycles used to fetch instructions and enqueue messages. The IRB effectively reduces the number of cycles needed to fetch instructions. The QRBs improve performance when processors are in non-idle states. Network analysis shows the likelihood of a processor being in an idle state when a message arrives from the network. Optimization of message handling by direct word enqueueing rather than the QRB's is recommended in this case.

Chapter 6

Conclusion

I'll note you in my book of memory.

— SHAKESPEARE, in *Henry IV, Part II, iv, 101*

This thesis reported the design and testing of the Memory Unit for the Message-Driven Processor (MDP). The memory organization provides hardware support for both indexed and set-associative access. Indexed access includes word access and row access. Row access was developed to increase the memory's bandwidth when enqueueing messages from the network or fetching instructions. The associative access provides an efficient method of translating virtual addresses into physical addresses.

The memory operates at 15.5MHZ. It uses two nonoverlapping clocks. The precharge clock is generated via self-timed precharge clock while decoding the memory address. A sense amplifier allows reading the memory in 9.3 ns. Writing the memory occurs in 10 ns. Comparators in the peripheral circuitry implement the set-associative operations.

Testing of a prototype memory chip verified the functionality of the address decoder. An error in the layout of the control circuitry (unplugged P-Well) prohibited routing data to the memory. A corrected chip will be sent for fabrication.

Possible Improvements

Several improvements of the memory design are possible. Circuit design improvements include sharing the precharge transistor and the column driver circuitry (see Section 3.2.4) allow an operating frequency of 16.67 MHz.

Improvements in the fabrication process could produce a faster and more compact memory. A buried contact in the RAM cell reduces the interconnect area, and accordingly, the cell area. Reducing the process's features size while maintaining the storage capacitance allows higher operating frequencies. Implementing error detection and correction circuit combats parity and soft error problems.

Further Research

Areas that will be further be pursued include completely verifying the functionality of the memory design and integrating the memory with the rest of the MDP Units on a single-chip to produce a fast processing node for the Jellybean Machine.

An architectural idea that deserves further researching is the MDP cache. Several parameters such as the cache size, associativity, TLB mapping algorithms, and replacement algorithms influence the performance. Real evaluation of the cache should be based on extensive trace-driven simulations with "real" workloads.

Appendix A

Register-Transfer Level Simulation of MU

This appendix contains the logic equations that are used in the Memory Unit's register-transfer level simulation. The equations refer to the Memory Unit Functional Block diagram shown in Figure 2.3. They are organized in 4 groups to be synchronized with the MDP's clock edges shown in Figure 2.2.

Phase 2, Falling Edge :

$Cycle_1$, $Cycle_2$, and $Cycle_3$ are generated to determine which cycle of the command is being executed.

$$
\begin{aligned}
Cycle_3 &= Cycle_2 \cdot (XLATE + PROBE) \\
Cycle_2 &= Cycle_1 \cdot (READ + XLATE + PROBE) \\
Cycle_1 &= Ready + \overline{Cycle_3} + \overline{Cycle_2} + \\
&\quad (\overline{Ready} \cdot (Refresh\_request + QRB0\_WRITE + QRB1\_WRITE))
\end{aligned}
$$

Phase 1, Rising Edge :

- $Qrb1\_request = QRB1\_WRITE \cdot \overline{Refresh\_request}$
- $Qrb0\_request = QRB0\_WRITE \cdot \overline{Refresh\_request} \cdot \overline{QRB1\_WRITE}$
- $Refresh\_delay = \overline{Cycle_1} \cdot Refresh\_request$
- $Qrb1\_empty = Cycle1 \cdot Qrb1\_request$
- If $(Qrb1\_empty)$ then $Local\_Row\_Buffer = QRB1$
- $Qrb0\_empty = Cycle1 \cdot Qrb0\_request$
- If $(Qrb0\_empty)$ then $Local\_Row\_Buffer = QRB0$
- If $(Cycle_1 \cdot (XLATE + PROBE + ENTER + PURGE))$ then $Key\_Regsiter = C\_bus$
- If $(Cycle_2 \cdot WRITE)$then $MDR = Local\_Row\_Buffer[c\_add]$
- If $(Cycle_2 \cdot READ \cdot \overline{squish})$then $Local\_Row\_Buffer[c\_add] = MDR$
- $Hit0 = Compare \cdot (Local\_Row\_Buffer[even\_word0] = Key\_Register)$
- $Hit1 = Compare \cdot (Local\_Row\_Buffer[even\_word1] = Key\_Register)$
- If $(Compare \cdot XLATE \cdot Hit0)$ then $MDR = Local\_Row\_Buffer[odd\_word0]$
- If $(Compare \cdot XLATE \cdot Hit1)$ then $MDR = Local\_Row\_Buffer[odd\_word1]$
- $MTrap = Compare \cdot XLATE \cdot \overline{Hit1} \cdot \overline{Hit0}$
- If $(Compare \cdot PROBE \cdot (Hit0 + Hit1)$ then $MDR = true$
- If $(Compare \cdot PROBE \cdot \overline{Hit1} \cdot \overline{Hit0})$ then $MDR = false$
- If $(Compare \cdot ENTER \cdot Hit0)$ then $Local\_Row\_Buffer[odd\_word0] = MDR$
- If $(Compare \cdot ENTER \cdot Hit1)$ then $Local\_Row\_Buffer[odd\_word1] = MDR$

- If $(Compare \cdot ENTER \cdot \overline{Hit1} \cdot \overline{Hit0})$ then $Local\_Row\_Buffer[odd\_word(random)] = Key\_Regsiter$ and $Local\_Row\_Buffer[even\_word(random)] = MDR$
- If $(Compare \cdot PURGE \cdot Hit0)$ then $Local\_Row\_Buffer[even\_word0] = nil$
- If $(Compare \cdot PURGE \cdot Hit1)$ then $Local\_Row\_Buffer[even\_word1] = nil$
- $\overline{Ready} = (Cycle1 \cdot (Refresh\_request + QRB0\_WRITE + QRB1\_WRITE + READ + XLATE + PROBE)) + (Cycle2 \cdot (XLATE + PROBE))$
- $MDR\_Valid = (Cycle2 \cdot \overline{squish} \cdot READ) + (Cycle3 \cdot \overline{squish} \cdot (XLATE + PROBE) \cdot (Hit0 + Hit1))$
- $Memory[R\_add] = Local\_Row\_Buffer$

Phase 1, Falling Edge:

- $Row\_add = (Memory\_Address \gg 2)$
- $Column\_add = Memory\_Address < 0 : 1 >$

Phase 2, Rising Edge:

- $Local\_Row\_Buffer = Memory[R\_add]$
- $Hit0 = 1$
- $Hit1 = 1$
- $MTrap = 0$
- $READY = 1$
- $Local\_Row\_Buffer = Memory[Row\_add]$
- $Refresh\_request = (refresh\_counter = 16) + Refresh\_delay)$

- If ($Refresh\_Counter = 16$) then $Refresh\_Counter = 0$
- If ($Refresh\_Counter \neq 16$) then $Refresh\_Counter = Refresh\_Counter + 1$
- If ($Qrb\_insert \cdot Priority\_1$)) then $QRB1[Qrb\_select] = IN\_Net$
- If ($Qrb\_insert \cdot Priority\_0$)) then $QRB0[Qrb\_select] = IN\_Net$
- If ($Cycle_1 \cdot READ$) then $Local\_Row\_Buffer[Column\_add] = MDR$
- If ($Cycle_1 \cdot (WRITE + ENTER)$) then $MDR = C\_bus$
- $Compare = Cycle_2 \cdot (XLATE + PROBE + ENTER + PURGE)$
- If ($Compare$) then $MDR = C\_bus$
- If (IRB_Load) then $IRB = Local\_Row\_Buffer$

Appendix B

Timing Diagram and Schematics of The MDP Memory

The table on the following page identifies the symbols and values for that figure. A detailed timing diagram of the memory array is shown in Figure B.1.

Figure B.2 is a slice through figure 3.2. Figure B.3 is the column select circuitry. Figures B.4 and B.5 are the control signals used in the memory.

Symbol	Definition	Delay (ns)
$t_{\varphi 1}$	phase 1	28.5
$t_{\varphi 2}$	phase 2	28.0
t_{ah}	address hold time	6.7
t_{psl}	$\overline{\varphi_{pre}}$ turn-on time	6.0
t_{ph}	$\overline{\varphi_{pre}}$ hold time	6.4
t_{psh}	$\overline{\varphi_{pre}}$ turn-off time	6.0
t_{rrs}	read row signal set-up time	5.0
t_{rrh}	read row signal hold time	23.0
t_{wrs}	write row signal set-up time	5.0
t_{wrh}	write row signal hold time	23.0
t_{cs}	compare set-up time	4.5
t_{ce}	compare evaluation	8.0
t_{cv}	compare result valid time	18.0
t_{ccss}	associative column-selection set-up time	19.5
t_{ccsh}	associative column-selection hold time	9.0
t_{rcss}	read column-selection set-up time	19.5
t_{rcsh}	read column-selection hold time	9.0
t_{wcss}	write column-selection set-up time	19.5
t_{wcsh}	write column-selection hold time	9.0

Table B.1: Memory Timing Table

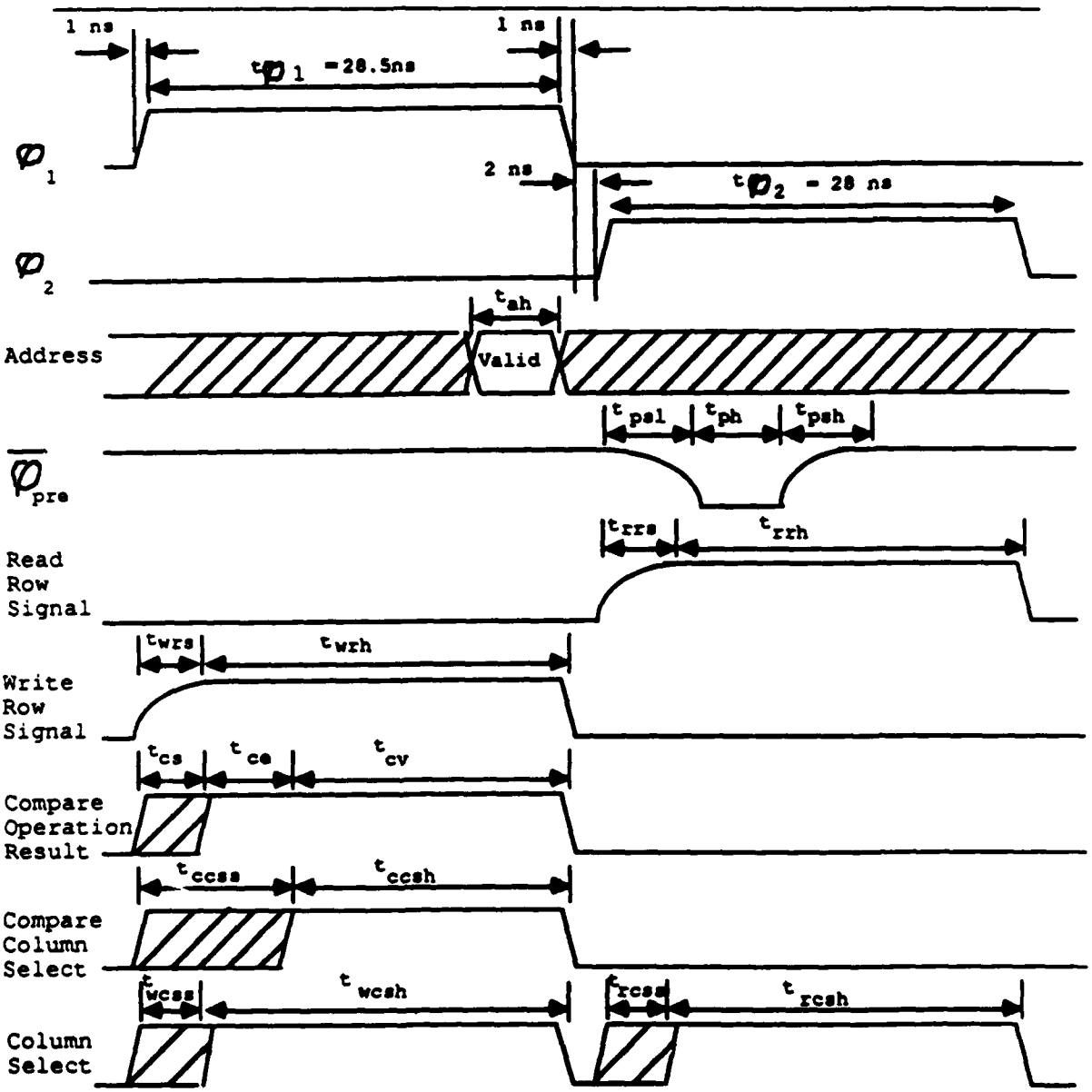


Figure B.1: Memory Timing

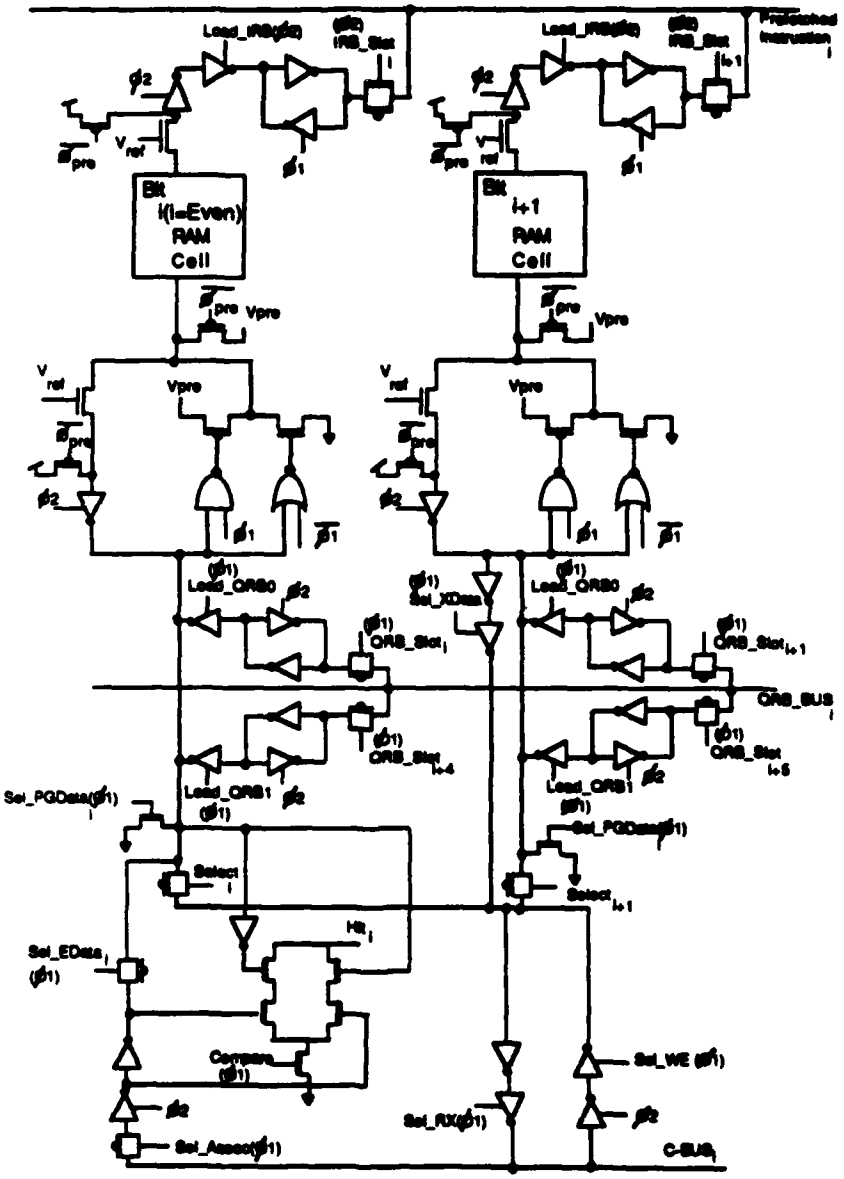


Figure B.2: A Slice in Figure 3.2

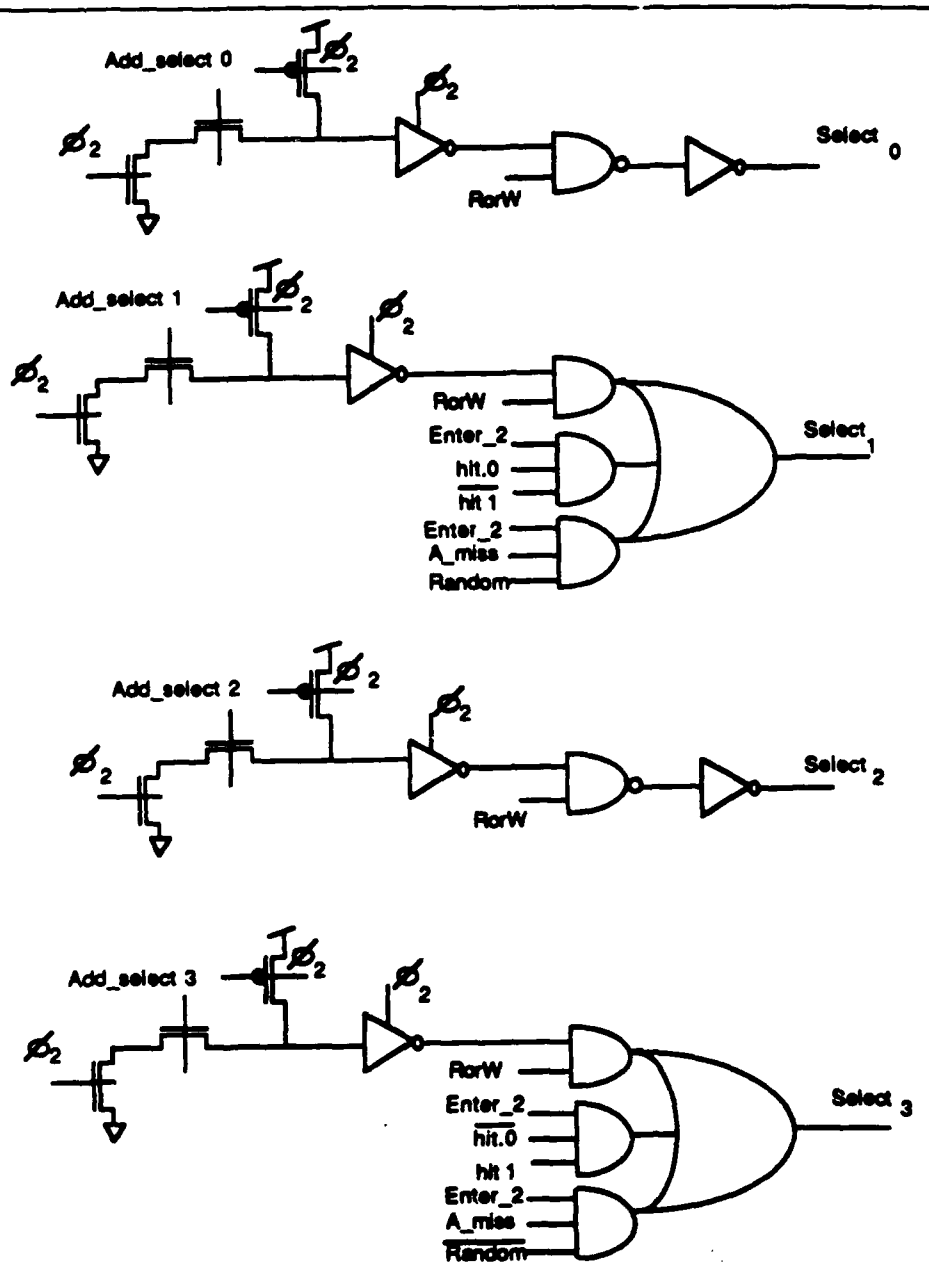


Figure B.3: Column Select Circuitry

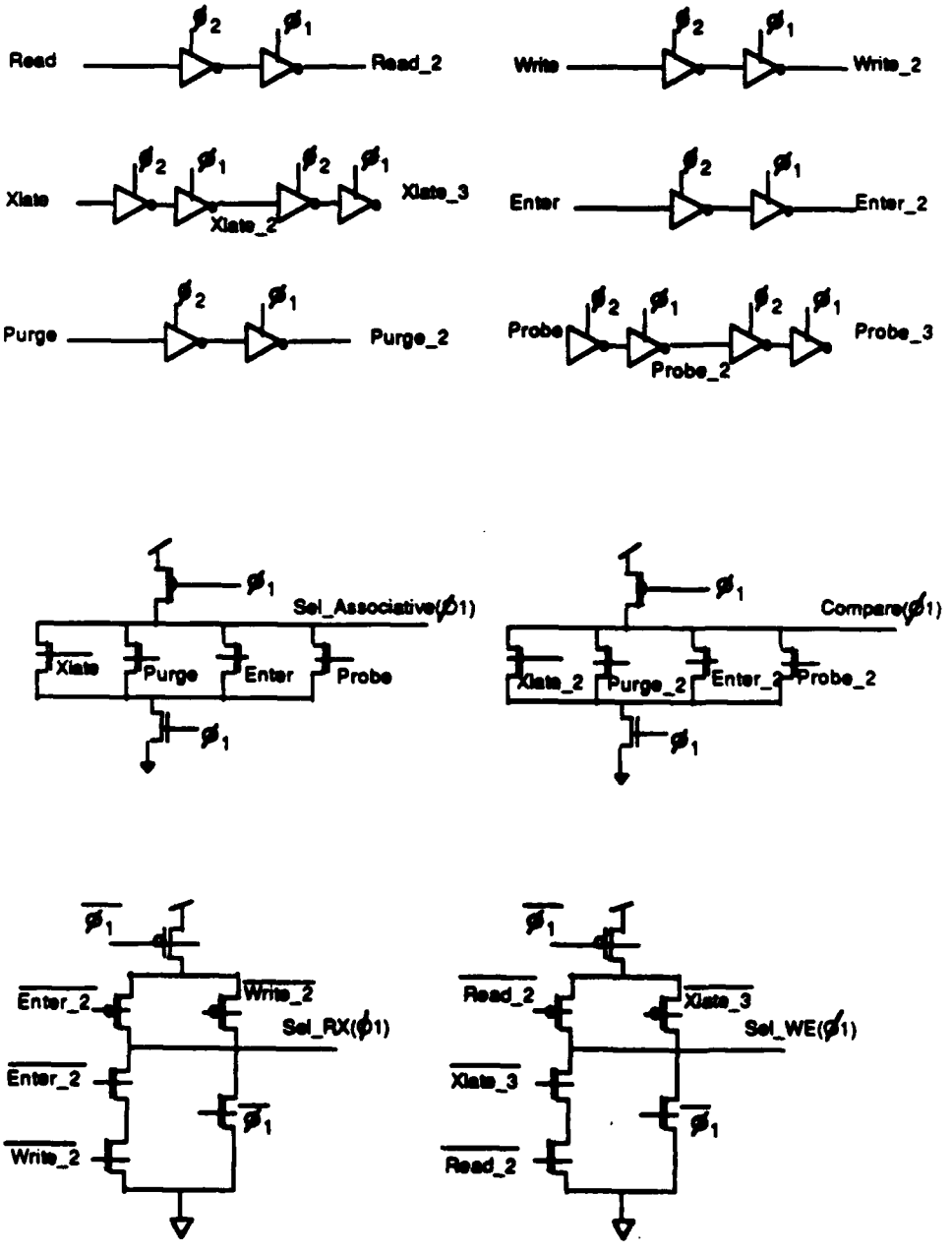


Figure B.4: Schematics of Memory Control Signals

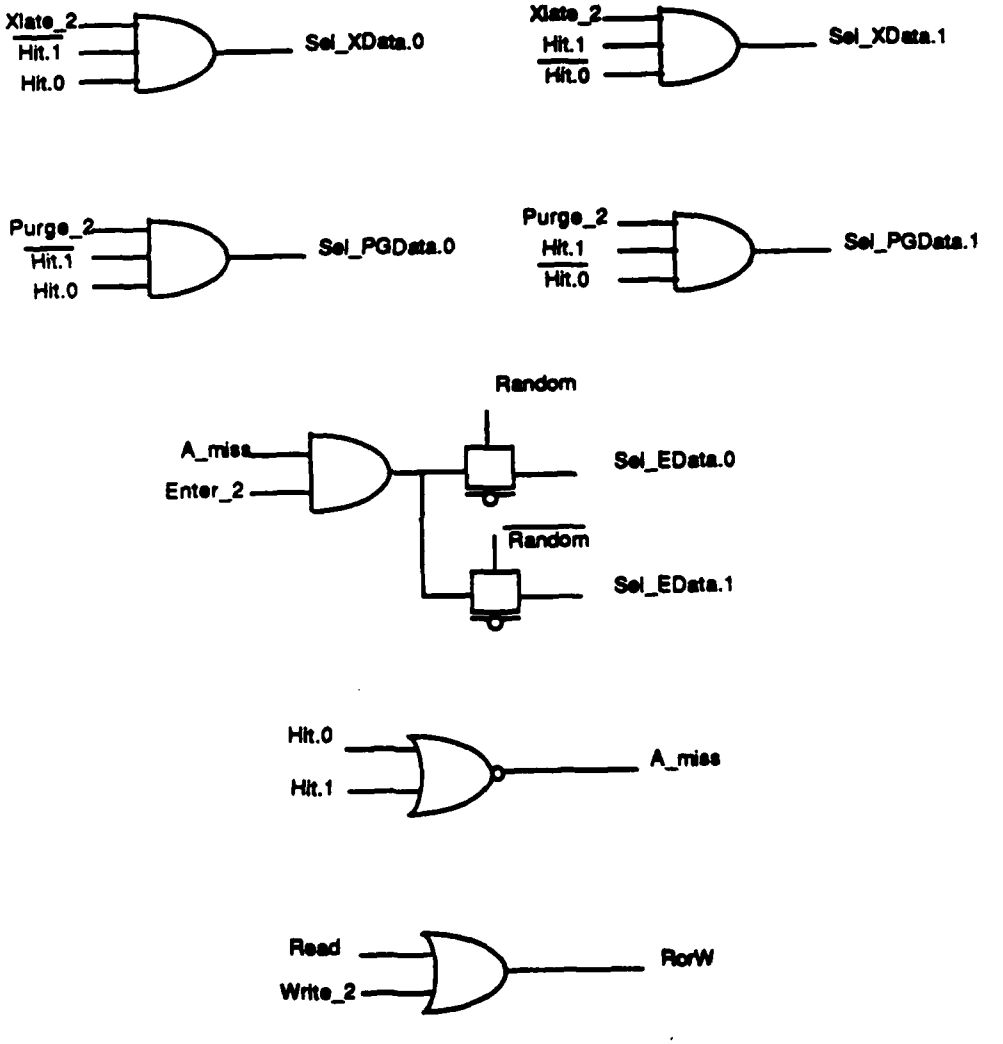


Figure B.5: ..and More Memory Control Signals

Appendix C

Electrical Parameters of a 2 μm CMOS Process

```
**********************************************************************
*                                  Slow Slow
**********************************************************************
.MODEL NSS NMOS LEVEL=3 RSH=0 TOX=275E-10 LD=.1E-6 XJ=.14E-6
+ CJ=1.6E-4 CJSW=1.8E-10 U0=550 VTO=1.022 CGSO=1.3E-10
+ CGDO=1.3E-10 NSUB=4E15 NFS=1E10
+ VMAX=12E4 PB=.7 MJ=.5 MJSW=.3 THETA=.06 KAPPA=.4 ETA=.14
.MODEL PSS PMOS LEVEL=3 RSH=0 TOX=275E-10 LD=.3E-6 XJ=.42E-6
+ CJ=7.7E-4 CJSW=5.4E-10 U0=180 VTO=-1.046 CGSO=4E-10
+ CGDO=4E-10 TPG=-1 NSUB=7E15 NFS=1E10
+ VMAX=12E4 PB=.7 MJ=.5 MJSW=.3 ETA=.06 THETA=.03 KAPPA=.4
*
* deltaLpoly = -.125um    deltaW = .9um   (one sided inward)
*
**********************************************************************
*                        Fast p-type  Slow n-type
**********************************************************************
.MODEL NFS NMOS LEVEL=3 RSH=0 TOX=250E-10 LD=.1E-6 XJ=.14E-6
+ CJ=1.6E-4 CJSW=1.5E-10 U0=550 VTO=1.03 CGSO=1.33E-10
+ CGDO=1.33E-10 NSUB=4E15 THETA=.06 KAPPA=.4 ETA=.14
```

```
+ VMAX=12E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
.MODEL PFS PMOS LEVEL=3 RSH=0 TOX=250E-10 LD=.4E-6 XJ=.6E-6
+ CJ=7E-4 CJSW=4.5E-10 U0=220 VT0=-.66 CGS0=5.5E-10
+ CGD0=5.5E-10 TPG=-1 NSUB=5E15 ETA=.06 THETA=.03 KAPPA=.4
+ VMAX=17E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
*
* deltaLpoly = 0um    deltaWp= .7um  deltaWn=.8um  (one sided inward)
*
*********************************************************************
*                        Fast p-type  Fast n-type
*********************************************************************
.MODEL NFF NMOS LEVEL=3 RSH=0 TOX=225E-10 LD=.15E-6 XJ=.21E-6
+ CJ=1.0E-4 CJSW=1.25E-10 U0=650 VT0=.628 CGS0=2.3E-10
+ CGD0=2.3E-10 NSUB=3E15 THETA=.06 KAPPA=.4 ETA=.14
+ VMAX=17E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
.MODEL PFF PMOS LEVEL=3 RSH=0 TOX=225E-10 LD=.4E-6 XJ=.6E-6
+ CJ=6E-4 CJSW=3.75E-10 U0=220 VT0=-.668 CGS0=6.2E-10
+ CGD0=6.2E-10 TPG=-1 NSUB=5E15 ETA=.06 THETA=.03 KAPPA=.4
+ VMAX=17E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
*
* deltaLpoly = .125um    deltaW = .6um   (one sided inward)
*
*********************************************************************
*                        Slow p-type  Fast n-type
*********************************************************************
.MODEL NSF NMOS LEVEL=3 RSH=0 TOX=250E-10 LD=.15E-6 XJ=.21E-6
+ CJ=1.0E-4 CJSW=1.5E-10 U0=650 VT0=.626 CGS0=2E-10
+ CGD0=2E-10 NSUB=3E15 THETA=.06 KAPPA=.4 ETA=.14
+ VMAX=17E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
.MODEL PSF PMOS LEVEL=3 RSH=0 TOX=250E-10 LD=.3E-6 XJ=.42E-6
+ CJ=7E-4 CJSW=4.5E-10 U0=180 VT0=-1.049 CGS0=4.2E-10
+ CGD0=4.2E-10 TPG=-1 NSUB=7E15 ETA=.06 THETA=.03 KAPPA=.4
+ VMAX=12E4 PB=.7 MJ=.5 MJSW=.3 NFS=1E10
*
```

```
* deltaLpoly = 0um     deltaWp = .8um  deltaWn=.7um (one sided inward)
*
*********************************************************************
```

Appendix D

Schematics for the Test Circuitry

This appendix contains schematics of the on-chip test circuitry. Figure D.1 is the voltage comparator.

Figure D.2 and Figure D.3 are a bit-slice of the pattern generator and its control circuitry. It consists of a random pattern generator, RPG, a linear shift register , LSR, a constant generator, a comparator and an output driver. The input to LSR's least significant bit is high when Reset_pattern is high, low otherwise. The input to the RPG's least significant bit is high when Reset_pattern is high. If the reset signal is low, the input is the exclusive-or of 5 register bits which generates 2^{34} different patterns. Signals Sel_A, Sel_B, Sel_C, Reset_Pattern, Shift_RPG, Shift_LSR, and Compare_Data are off-chip inputs. The result of the data comparison is an off-chip output.

Figure D.4 and Figure D.5 are a bit-slice of the address generator and its control circuitry. It consists of an incrementer, a register, and a comparator. Inputs Reset_Address_Generator, Enable_Address_Register, Increment_Add_Generator, and Compare_Address are off-chip signals. The carry out of the most significant address bit and the result of the address comparison are connected to the outside world.

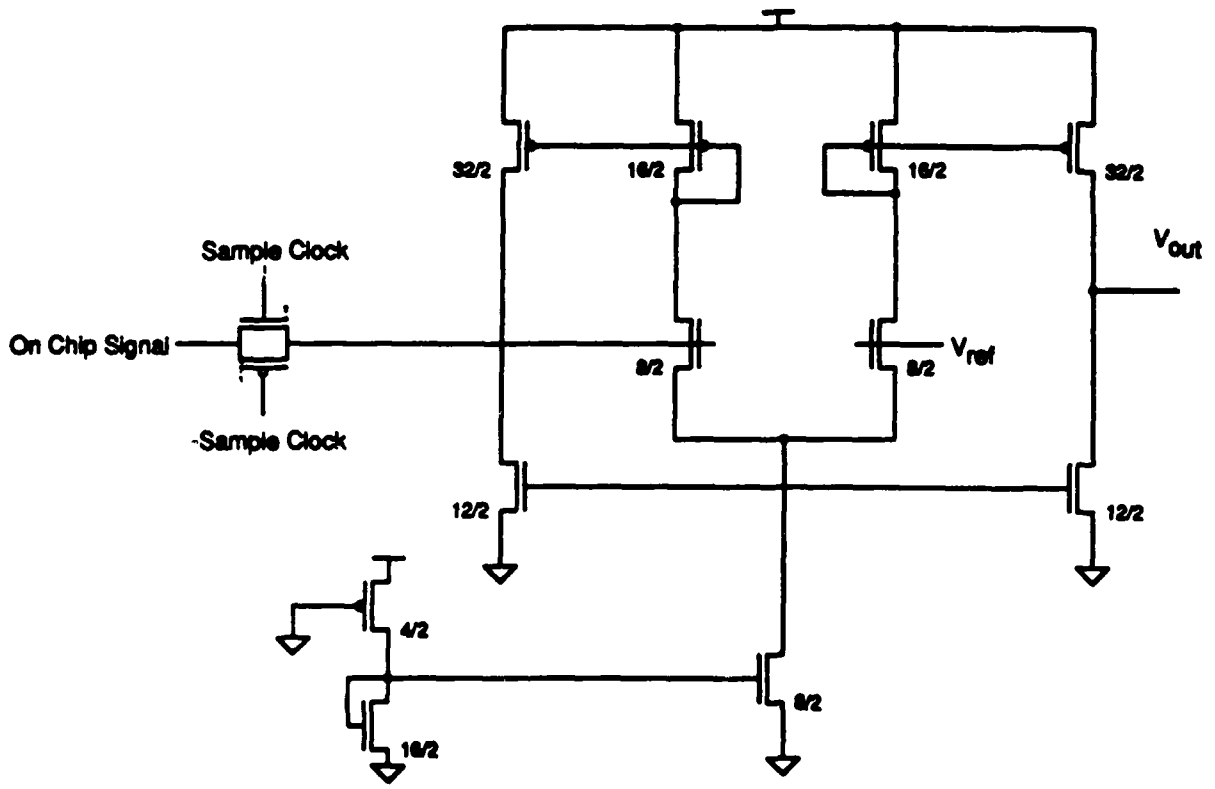


Figure D.1: Voltage Comparator

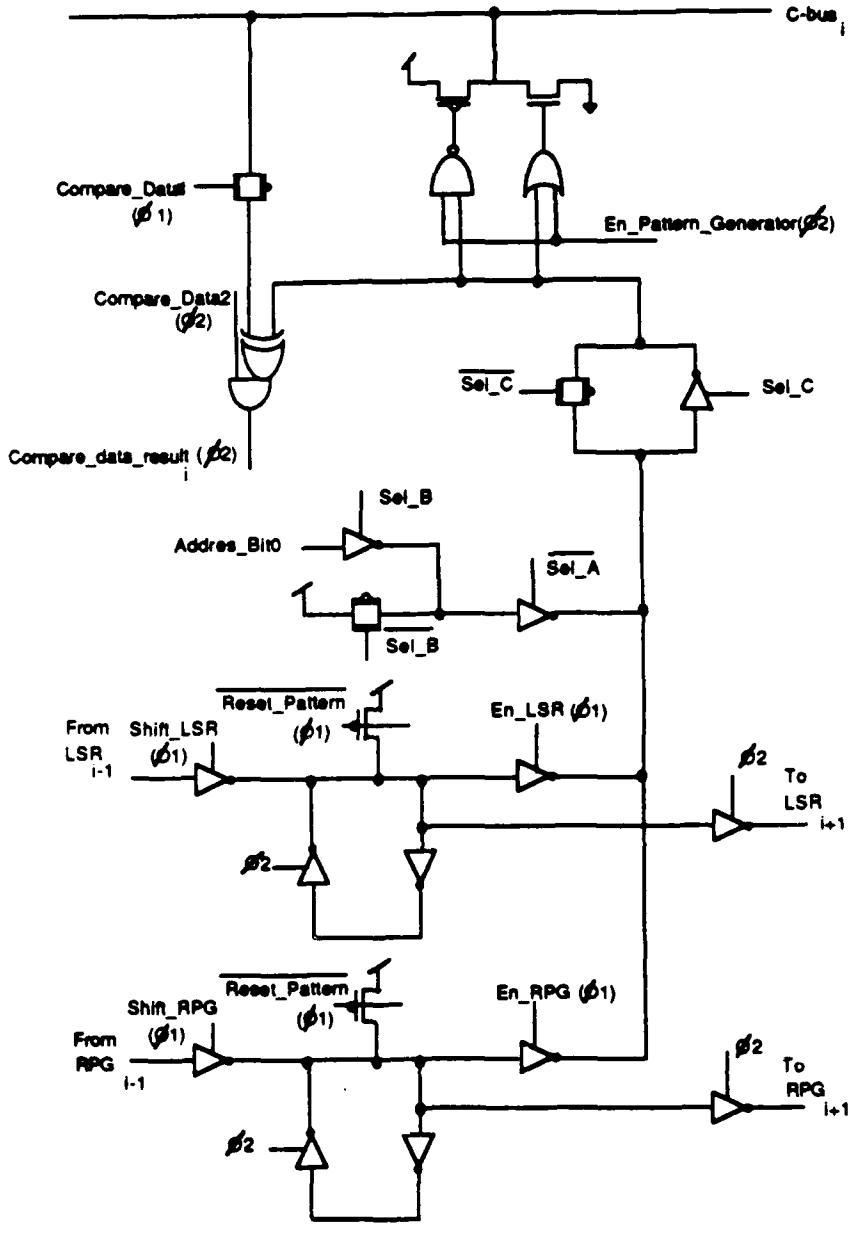


Figure D.2: Pattern Generator

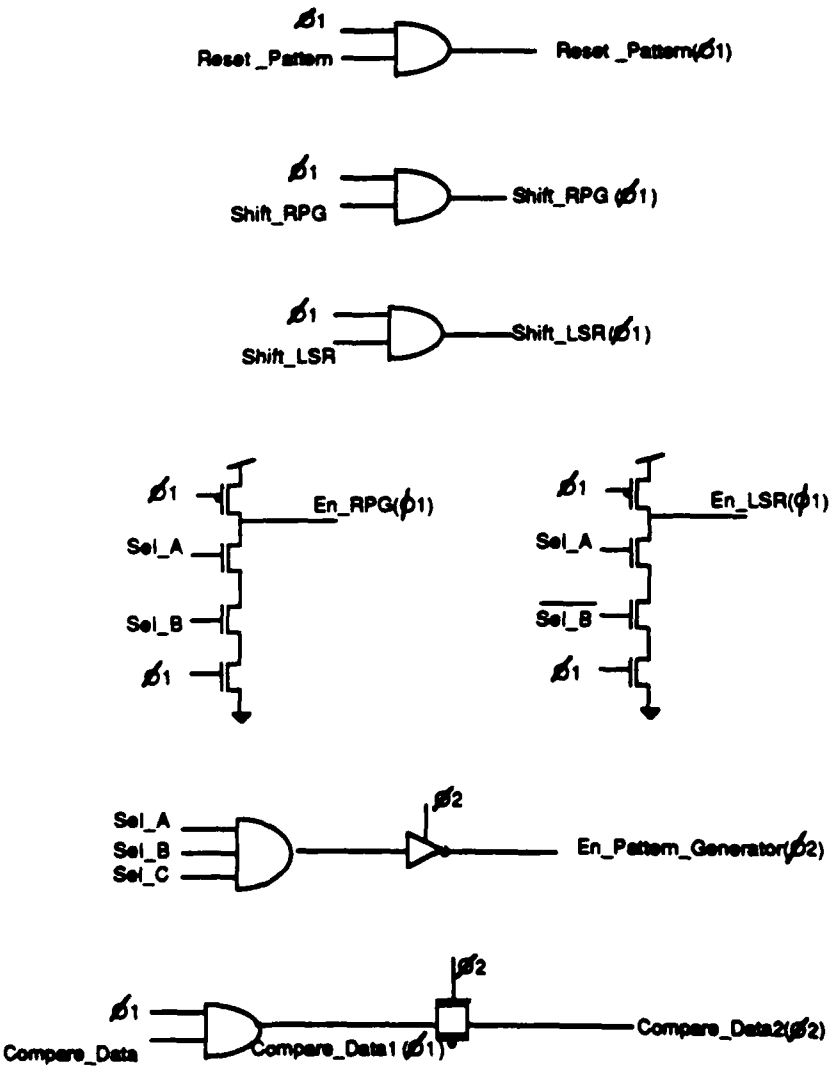


Figure D.3: Pattern Generator Control Signals

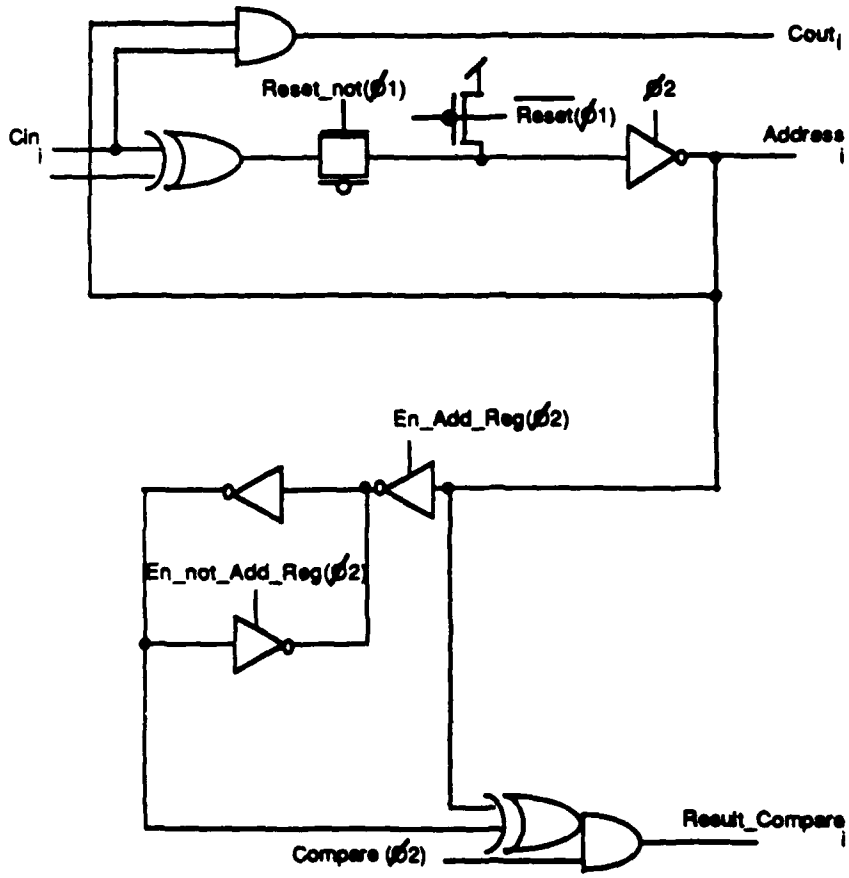


Figure D.4: Address Generator

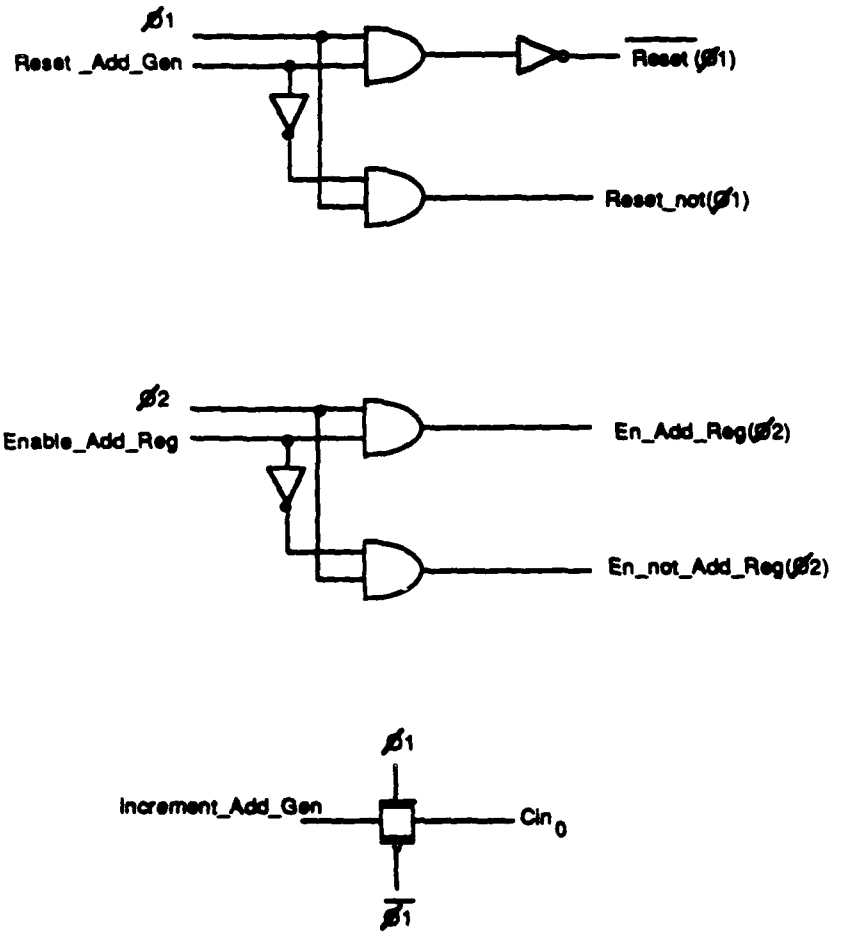


Figure D.5: Control Circuit for Address Generator

Bibliography

[BCP62] L. Bloom, M. Cohen, and S. Proter. Considerations in the Design of a Computer with High Logic-to-Memory Speed Rates. *Proceedings of Sessions on Gigacycle Computing Systems*, 53–63, January 1962.

[BF76] Melvin a. Breuer and Arthur D. Friedman. *Diagnosis and Reliable Design of Digital Systems*, pages 139–160. Computer Science Press, Inc., 1976.

[Bre88] Silvano Brewster. *Probabilistic Analysis of Soft Errors in VLSI Circuits.* PhD thesis, Massachusetts Institute Technology, June 1988.

[D*87] W. J. Dally et al. Architecture of a Message-Driven Processor. In *Proceeding of the 14th Annual Symposium of Computer Architecture*, June 1987.

[Dal] W. J. Dally. The J-Machine. Darpa Report, 1987.

[DG85] Daniel W. Dobberpuhl and Lance A. Glasser. *The Design and Analysis of VLSI Circuits*, pages 272–3. Addison-Wesley, 1985.

[DS87] W. J. Dally and Paul Y. Song. Design of a Self-Timed VLSI Multicomputer Communication Controller. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 230–4, October 1987.

[FD] S. Fiske and W. J. Dally. The Reconfigurable Arithmetic Processor. To be presented in the 15th Annual Symposium of Computer Architecture.

[I*88] Michihiro Inoue et al. A 16Mb DRAM with An Open Bit-Line Architecture. In *IEEE International Solid-State Circuits Conference*, page 246, February 1988.

[MW79] Timothy C. May and Marray H. Woods. Alpha-Particle-Induced Soft Errors in Dynamic Memories. In *IEEE Transactions on Electron Devices*, pages 2–9, Jan 1979.

[P*81] E. W. Pugh et al. Solid State Memory Development in IBM. *IBM Journal of Research and Development*, 25(5):585–602, September 1981.

[Son88] Paul Song. *Design of A network for Concurrent Message Passing Systems.* Master's thesis, Massachusetts Institute Technology, May 1988.

[Zip84] Richard Zippel. *A Survey of Memory Design Techniques, Draft.* 1984.